剑指 Offer 13. 机器人的运动范围

```
var visited [][]bool
var count int
func movingCount(m int, n int, k int) int {
    visited = make([][]bool, m)
    count = 0
    for i, _ := range visited {
        visited[i] = make([]bool, n)
    }
    dfs(0,0, m, n, k)
    return count
}

func dfs(i, j, m, n, k int) {
    visited[i][j] = true
    count++
    directions := [][]int{{-1,0}, {1,0}, {0,-1}, {0,1}}
    for di := 0; di < 4; di++ {
        newi := i + directions[di][0]
        newj := j + directions[di][1]
        if newi >= m || newi < 0 || newj >= n || newj < 0 ||
            visited[newi][newj] == true ||
            check(newi, newj, k) == false {
            continue
        }
        dfs(newi, newj, m, n, k)
    }
}

func check(i, j, k int) bool{
    sum := 0
    for i != 0 {
        sum += (i%10)
        i /= 10
    }
    for j != 0 {
        sum += (j%10)
        j /= 10
    }
    return sum <= k
}
```

面试题 08.10. 颜色填充

```
func floodFill(image [][]int, sr int, sc int, newColor int) [][]int {
    n := len(image)
    m := len(image[0])
    dfs(image, n, m, sr, sc, image[sr][sc], newColor)
    return image
}

func dfs(image [][]int, n, m, sr, sc, color, newColor int) {
    image[sr][sc] = newColor
    dirs := [][]int{{-1,0},{1,0},{0,1},{0,-1}}
    for k := 0; k < 4; k++ {
        newr := sr + dirs[k][0]
        newc := sc + dirs[k][1]
        if newr < 0 || newr >= n || newc < 0 || newc >= m || image[newr][newc] !=
color ||
            image[newr][newc] == newColor {
            continue
        }
        dfs(image, n, m, newr, newc, color, newColor)
    }
}
```

面试题 04.01. 节点间通路

```
var visited []bool
var adj []map[int]bool
var found bool
func findWhetherExistsPath(n int, graph [][]int, start int, target int) bool {
    visited = make([]bool, n)
    adj = make([]map[int]bool, n)
    found = false
    for i := 0; i < n; i++ {
        adj[i] = make(map[int]bool, 0)
    }
    for i := 0; i < n; i++ {
        if !adj[graph[i][0]][graph[i][1]] {
            adj[graph[i][0]][graph[i][1]] = true
        }
    }
    dfs(start, target)
    return found
}

func dfs(cur, target int) {
    if found {return}
    if cur == target {
        found = true
        return
    }
    visited[cur] = true
    for next, _ := range adj[cur] {
        if !visited[next] {
            dfs(next, target)
        }
    }
}
```

200. 岛屿数量

```
var visited [][]bool
var h int
var w int
func numIslands(grid [][]byte) int {
    h = len(grid)
    w = len(grid[0])
    visited = make([][]bool, h)
    for i := 0; i < h; i++ {
        visited[i] = make([]bool, w)
    }
    result := 0
    for i := 0; i < h; i++ {
        for j := 0; j < w; j++ {
            if visited[i][j] != true && grid[i][j] == '1' {
                result++
                dfs(grid, i, j)
            }
        }
    }
    return result
}

func dfs(grid [][]byte, i, j int) {
    directions := [][]int{{-1,0},{1,0},{0,-1},{0,1}}
    visited[i][j] = true
    for k := 0; k < 4; k++ {
        newi := i + directions[k][0]
        newj := j + directions[k][1]
        if newi >= 0 && newi < h && newj >= 0 && newj < w &&
            visited[newi][newj] == false && grid[newi][newj] == '1' {
            dfs(grid, newi, newj)
        }
    }
}
```

面试题 16.19. 水域大小

```
var count int
var n int
var m int
func pondSizes(land [][]int) []int {
   n = len(land)
   m = len(land[0])
   result := make([]int, 0)
   for i := 0; i < n; i++ {
       for j := 0; j < m; j++ {
           if land[i][j] == 0 {
               count = 0
               dfs(land, i, j)
               result = append(result, count)
           }
       }
   }
   sort.Ints(result)
   return result
}

func dfs(land [][]int, curi, curj int) {
    count++
    land[curi][curj] = 1
    dirs := [][]int{{-1,0},{1,0},{0,1},{0,-1},
        {-1,-1},{1,1},{-1,1},{1,-1}}
    for d := 0; d < 8; d++ {
        newi := curi + dirs[d][0]
        newj := curj + dirs[d][1]
        if newi >= 0 && newi < n && newj >= 0 && newj < m &&
            land[newi][newj] == 0 {
            dfs(land, newi, newj)
        }
    }
}
```

207. 课程表

```
func canFinish(numCourses int, prerequisites [][]int) bool {
    adjs := make([][]int, numCourses)
    for i := 0; i < numCourses; i++ {
        adjs[i] = make([]int, 0)
    }
    indegrees := make([]int, numCourses)
    for i := 0; i < len(prerequisites); i++ {
        adjs[prerequisites[i][1]] = append(adjs[prerequisites[i][1]],
prerequisites[i][0])
        indegrees[prerequisites[i][0]]++
    }
    zeroInDegrees := make([]int, 0)
    for i := 0; i < len(indegrees); i++ {
        if indegrees[i] == 0 {
            zeroInDegrees = append(zeroInDegrees, i)
        }
    }
    zeroInDegreesCount := 0
    for len(zeroInDegrees) > 0 {
        coursei := zeroInDegrees[0]
        zeroInDegrees = zeroInDegrees[1:]
        zeroInDegreesCount++
        for _, coursej := range adjs[coursei] {
            indegrees[coursej]--
            if indegrees[coursej] == 0 {
                zeroInDegrees = append(zeroInDegrees, coursej)
            }
        }
    }
    return zeroInDegreesCount == numCourses
}
```

79. 单词搜索

```
var existed bool
var h int
var w int
func exist(board [][]byte, word string) bool {
    h = len(board)
    w = len(board[0])
    existed = false
    for i := 0; i < h; i++ {
        for j := 0; j < w; j++ {
            visited := make([][]bool, h)
            for k := 0; k < len(visited); k++ {
                visited[k] = make([]bool, w)
            }
            dfs(board, word, i, j, 0, visited)
            if existed {return true}
        }
    }
    return false
}

func dfs(board [][]byte, word string, i, j, k int, visited [][]bool) {
    if existed == true {return}
    if word[k] != board[i][j] {
        return
    }
    visited[i][j] = true
    if k == len(word)-1 {
        existed = true
        return
```

```go
    }
    directions := [][]int{{-1,0},{1,0},{0,-1},{0,1}}
    for di := 0; di < 4; di++ {
        nexti := i + directions[di][0]
        nextj := j + directions[di][1]
        if nexti >= 0 && nexti < h && nextj >= 0 && nextj < w &&
            !visited[nexti][nextj] {
            dfs(board, word, nexti, nextj, k+1, visited)
        }
    }
    visited[i][j] = false
}
```

1306. 跳跃游戏 III

```go
var visited []bool
var reached bool
func canReach(arr []int, start int) bool {
    n := len(arr)
    visited = make([]bool, n)
    reached = false
    dfs(arr, start)
    return reached
}

func dfs(arr []int, curi int) {
    if reached {return}
    if arr[curi] == 0 {
        reached = true
        return
    }
    visited[curi] = true
    move2left := curi - arr[curi]
    if move2left >= 0 && move2left < len(arr) &&
        visited[move2left] == false {
        dfs(arr, move2left)
    }
    move2right := curi + arr[curi]
    if move2right >= 0 && move2right < len(arr) &&
        visited[move2right] == false {
        dfs(arr, move2right)
    }
}
```

752. 打开转盘锁

```go
func openLock(deadends []string, target string) int {
    deadset := make(map[string]bool, 0)
    for _, d := range deadends {
        deadset[d] = true
    }
    if deadset["0000"] {return -1}
    queue := make([]string, 0)
    visited := make(map[string]bool,0)
    queue = append(queue, "0000")
    visited["0000"] = true
    depth := 0
    for len(queue) > 0 {
        size := len(queue)
        k := 0
        for k < size {
            node := queue[0]
            queue = queue[1:]
            k++
            if node == target {
                return depth
            }
            newnodes := genNewNode(node)
            for _, newnode := range newnodes {
                if visited[newnode] || deadset[newnode] {
                    continue
                }
                queue = append(queue, newnode)
                visited[newnode] = true
            }
        }
        depth++
    }
    return -1
}

func genNewNode(node string) []string {
    newnodes := make([]string, 0)
    change := []int{-1, 1}
    for i := 0; i < 4; i++ {
        for k := 0; k < 2; k++ {
            newNode := make([]byte, 4)
            for j := 0; j < i; j++ {
                newNode[j] = node[j]
            }
            for j := i+1; j < 4; j++ {
                newNode[j] = node[j]
            }
            newC := fmt.Sprintf("%d", (int(node[i]-'0') + change[k] + 10) % 10)
            newNode[i] = newC[0]
            newnodes = append(newnodes, string(newNode))
        }
    }
    return newnodes
}
```

面试题 17.22. 单词转换

```
var visited map[string]bool
var resultPath []string
var found bool
func findLadders(beginWord string, endWord string, wordList []string) []string {
    visited = make(map[string]bool, 0)
    resultPath = make([]string, 0)
    found = false
    dfs(beginWord, endWord, []string{}, wordList)
    return resultPath
}

func dfs(curWord, endWord string, path []string, wordList []string) {
    if found {return}
    path = append(path, curWord)
    visited[curWord] = true
    if curWord == endWord {
        resultPath = append(resultPath, path...)
        found = true
```

```go
        return
    }
    for i := 0; i < len(wordList); i++ {
        nextWord := wordList[i]
        if visited[nextWord] || !isValidChange(curWord, nextWord) {
            continue
        }
        dfs(nextWord, endWord, path, wordList)
    }
    path = path[:len(path)-1]
}

func isValidChange(word1, word2 string) bool{
    diff := 0
    for i := 0; i < len(word1); i++ {
        if word1[i] != word2[i] {
            diff++
        }
    }
    return diff == 1
}
```