Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. І. Сікорського»

Кафедра інженерії програмного забезпечення в енергетиці

# Лабораторна робота № 1
з курсу: «Методологія інженерії програмного забезпечення»

**Виконав:**
студент 1-го курсу магістратури,
групи ТВ-31мн
Черноусов Денис Ігорович

**Перевірив:**
Шпурик В.В.

Київ 2024

**Завдання:**

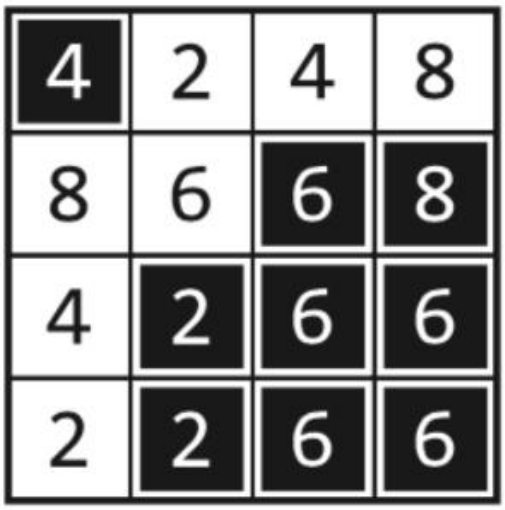
Завдання 26: Розробіть алгоритм вирішення задачі та реалізуйте його у вигляді програми мовою ANSI C++. Зафарбуйте деякі клітини так, щоб у кожному рядку або стовпці не було чисел, що повторюються. Зафарбовані клітини не повинні стикатися одна з одною. Усі незафарбовані клітини повинні з'єднуватися одна з одною сторонами по горизонталі або по вертикалі так, щоб вийшов єдиний безперервний простір із незафарбованих клітин.

Ваше завдання. Зафарбовані клітини можуть стикатися одна з одною. У кожному рядку та кожному стовпці повинна залишатися принаймні одна незафарбована клітина.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 4 | 3 | 4 | 1 | 3 | 2 | 2 |
| 1 | 1 | 2 | 3 | 2 | 1 | 3 | 2 | 2 |
| 3 | 2 | 1 | 4 | 3 | 3 | 2 | 1 | 3 |
| 4 | 3 | 4 | 2 | 3 | 1 | 1 | 2 | 4 |
| 4 | 2 | 1 | 1 | 2 | 3 | 3 | 4 | 1 |
| 2 | 2 | 3 | 3 | 4 | 4 | 4 | 1 | 2 |
| 2 | 3 | 3 | 1 | 3 | 2 | 2 | 4 | 1 |
| 4 | 4 | 2 | 1 | 3 | 1 | 2 | 3 | 3 |
| 4 | 4 | 2 | 1 | 1 | 1 | 2 | 3 | 3 |

Приклад та його вирішення:

| | | | |
|---|---|---|---|
| 4 | 2 | 4 | 8 |
| 8 | 6 | 6 | 8 |
| 4 | 2 | 6 | 6 |
| 2 | 2 | 6 | 6 |




**Опис алгоритму:**

Примітки:

Мова виконання - GoLang

Стіни - це крайні рядки або стовпці двовимірної матриці.

З самого початку усі клітинки зафарбовані

Виходячи з завдання можна вивести такі **властивості**:

1) Біля кожної стіни має бути принаймні одна незафарбована клітина.

2) Оскільки має бути безперервний простір для незафарбованих клітин, то клітини, з властивості 1), мають мати принаймні 1 маршрут з незафарбованих клітин за яким вони з’єднуються.

3) З властивості 2) знаємо, що мають бути такі маршрут, що веде зверху в низ та зправа наліво. Очевидно, що ці маршрути перетинаються.

*4)* Якщо з’єднати 2 маршрути, то необов’яково виконується правило: *“Зафарбуйте деякі клітини так, щоб у кожному рядку або стовпці не було чисел, що повторюються.”*.

З властивостей описаних вище випливає **глобальний алгоритм, що мітить такі глобальні кроки**:

1) Знайти усі можливі шляхи з верхньої стіни до нижньої стіни, тобто масив матриць

2) Знайти усі можливі шляхи з лівої стіни до правої стіни, тобто масив матриць

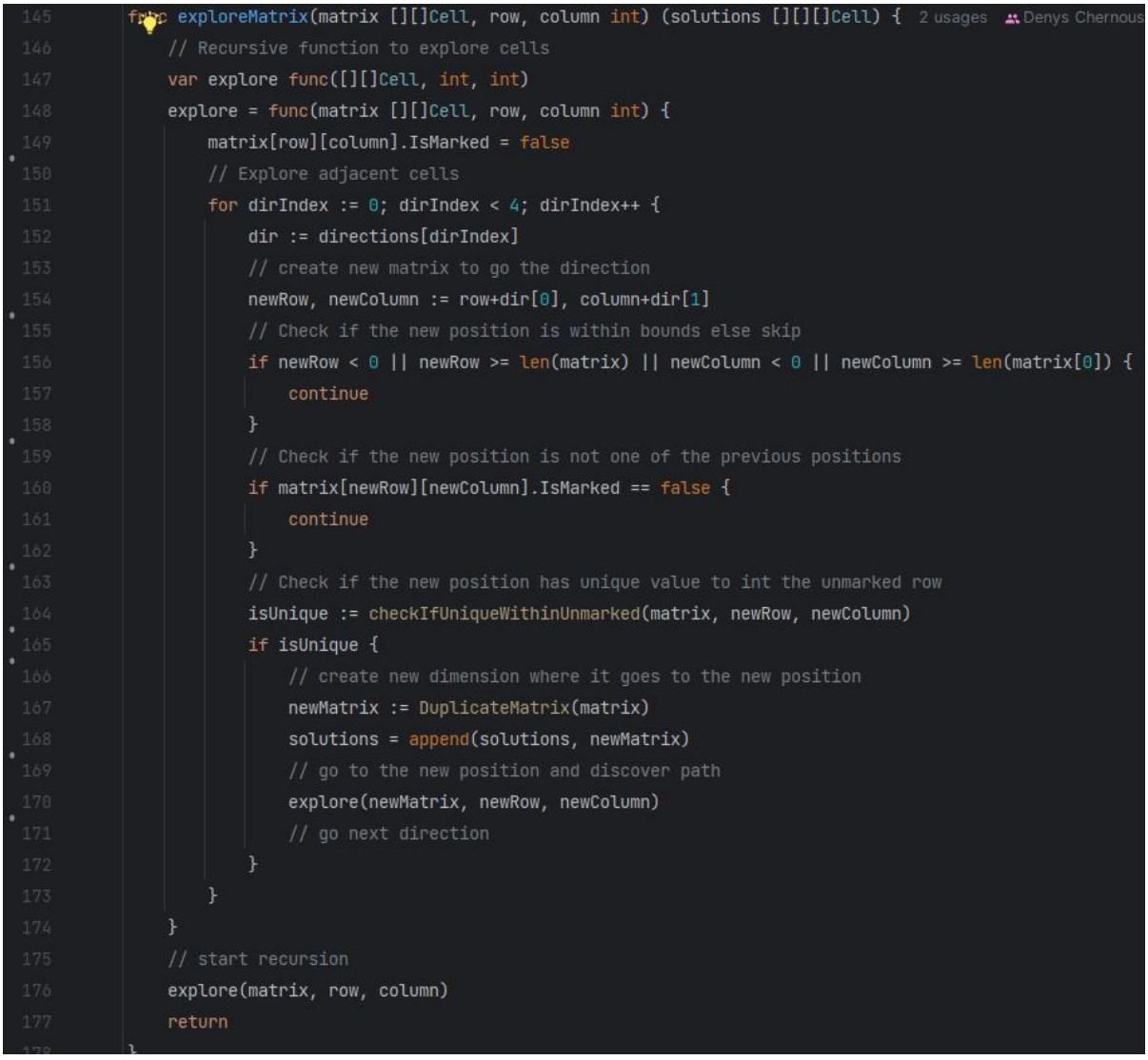
3) Комбінуючи усі шляхи з пункту 1) та усі шляхи з пункту 2), надати як вирішення задачі першу комбінацію, що задовільняє правило: *“Зафарбуйте деякі клітини так, щоб у кожному рядку або стовпці не було чисел, що повторюються.”*.

Програмно це виглядатиме ось так:

```go
////////////////////////////////////////////////////////////////////////////////////////////////////
// Solve by pathfinding and combining 2 paths

func SolveMatrix(matrix [][]Cell) [][]Cell {  2 usages  Denys Chernousov *

    // make way down from the top
    var solutionsDown [][][]Cell
    for i := 0; i < len(matrix[0]); i++ {
        solutionsDown = append(solutionsDown, makeWayDown(DuplicateMatrix(matrix),  initRow: 0, i)...)
    }

    // make way from the left to the right
    var solutionsRight [][][]Cell
    for i := 0; i < len(matrix); i++ {
        solutionsRight = append(solutionsRight, makeWayRight(DuplicateMatrix(matrix), i,  initColumn: 0)...)
    }

    // combine the ways and check if it doesn't break the rules
    for _, solutionDown := range solutionsDown {
        for _, solutionRight := range solutionsRight {
            combined := combineMatrixes(solutionDown, solutionRight)
            if iterateMatrixAndCheckIfGood(combined) {
                return combined
            }
        }
    }

    return nil
}
```

Розглянемо кожен з кроків детальніше. Почнемо з глобального кроку 1:

1)



```
func exploreMatrix(matrix [][]Cell, row, column int) (solutions [][][]Cell) {  2 usages  Denys Chernous
    // Recursive function to explore cells
    var explore func([][]Cell, int, int)
    explore = func(matrix [][]Cell, row, column int) {
        matrix[row][column].IsMarked = false
        // Explore adjacent cells
        for dirIndex := 0; dirIndex < 4; dirIndex++ {
            dir := directions[dirIndex]
            // create new matrix to go the direction
            newRow, newColumn := row+dir[0], column+dir[1]
            // Check if the new position is within bounds else skip
            if newRow < 0 || newRow >= len(matrix) || newColumn < 0 || newColumn >= len(matrix[0]) {
                continue
            }
            // Check if the new position is not one of the previous positions
            if matrix[newRow][newColumn].IsMarked == false {
                continue
            }
            // Check if the new position has unique value to int the unmarked row
            isUnique := checkIfUniqueWithinUnmarked(matrix, newRow, newColumn)
            if isUnique {
                // create new dimension where it goes to the new position
                newMatrix := DuplicateMatrix(matrix)
                solutions = append(solutions, newMatrix)
                // go to the new position and discover path
                explore(newMatrix, newRow, newColumn)
                // go next direction
            }
        }
    }
    // start recursion
    explore(matrix, row, column)
    return
}
```

Ініціалізуємо масив матриць та запускає рекурсію з верхньої стіни

2) Відбувається прохід в можливі напрямки: ліво, низ, вправо, вверх - ща допомогою циклу, проте якщо нові координати вказують за межі метриці, або на розмарковане значення, то відбувається пропуск напрямку.

3) Кожен раз коли клітинку можливо розмаркувати згідно правилу “Зафарбуйте деякі клітини так, щоб у кожному рядку або стовпці не було чисел, що повторюються.”, що перевіряє функція

checkIfUniqueWithinUnmarked, існуюча матриця копіюється в масив матриць та відбувається рекурсія. За допомогою цього можливо перебрати усі варіанти маршрутів та записати їх в масив матриць.

4)
```
118  func makeWayDown(originMatrix [][]Cell, initRow, initColumn int) (result [][][]Cell) {
119      solutions := exploreMatrix(originMatrix, initRow, initColumn)
120      for _, solution := range solutions {
121          if checkIfTouchesBottomWall(solution) {
122              result = append(result, solution)
123          }
124      }
125      return
126  }
```

Вертаємо матриці, що мають незафарбовану клітинку, яка торкається нижньої стіни.

Глобальний крок 2 подібний до 1го, проте має інші початкові позиції (біля лівої стіни) та вертає матриці, що торкаються правої стіни.

```
128  func makeWayRight(originMatrix [][]Cell, initRow, initColumn int) (result [][][]Cell) {
129      solutions := exploreMatrix(originMatrix, initRow, initColumn)
130      for _, solution := range solutions {
131          if checkIfTouchesRightWall(solution) {
132              result = append(result, solution)
133          }
134      }
135      return
136  }
```

Третій глобальний крок:

1)
```go
func combineMatrixes(matrix1 [][]Cell, matrix2 [][]Cell) [][]Cell {
    combined := DuplicateMatrix(matrix1)
    for i := 0; i < len(matrix1); i++ {
        for j := 0; j < len(matrix1[0]); j++ {
            if matrix2[i][j].IsMarked == false {
                combined[i][j].IsMarked = false
            }
        }
    }
    return combined
}
```

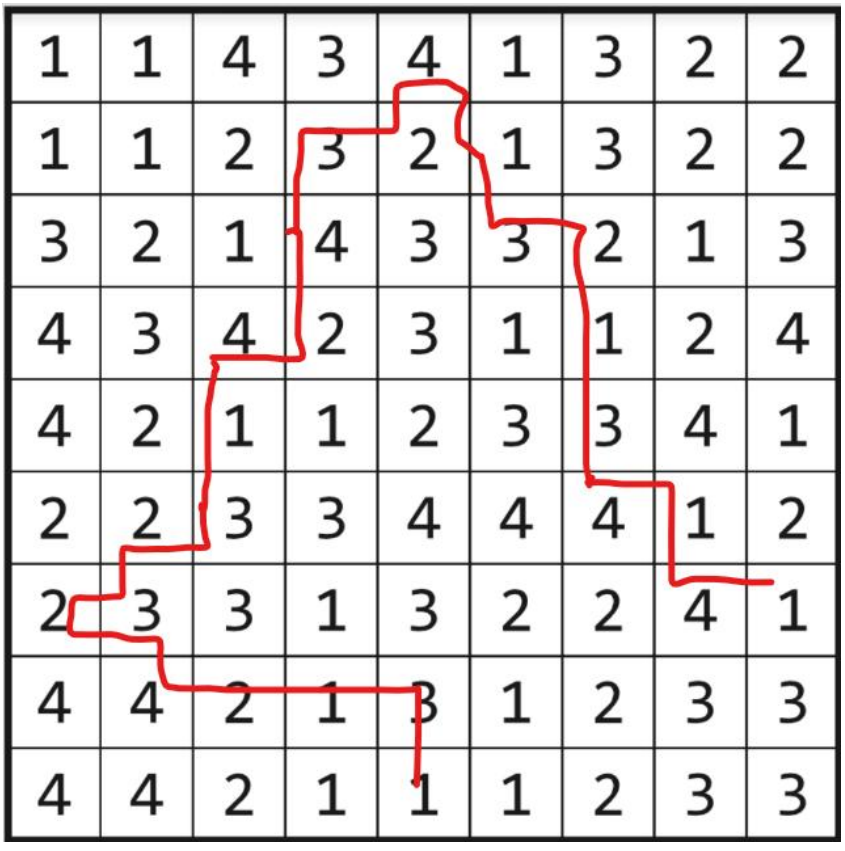
Комбінуємо матриці, тобто створюємо третю матрицю, де переносимо всі невідмічені клітинки

2)
```go
func iterateMatrixAndCheckIfGood(matrix [][]Cell) bool {  1 usage  Denys Chernousov *

    // find first unmarked
    var startCol int
    for i := 0; i < len(matrix[0]); i++ {...}

    isGood := true
    visited := make(map[[2]int]bool) // Keep track of visited cells to avoid revisiting
    var iterate func(int, int)
    iterate = func(row int, col int) {
        if row < 0 || col < 0 || row >= len(matrix) || col >= len(matrix[0]) ||
            visited[[2]int{row, col}] || matrix[row][col].IsMarked == true { ... }
        // Mark the cell as visited
        visited[[2]int{row, col}] = true
        // Perform check operation
        if checkIfUniqueWithinUnmarked(matrix, row, col) == false {
            isGood = false
            return
        }
        // Recursively call iterate for neighboring cells
        iterate(row+1, col) // Right
        iterate(row-1, col) // Left
        iterate(row, col+1) // Down
        iterate(row, col-1) // Up

    }
    iterate(0, startCol)
    return isGood
}
```

Знаходимо будь-яку невідмічену клітинку в першому рядку. Починаючи з

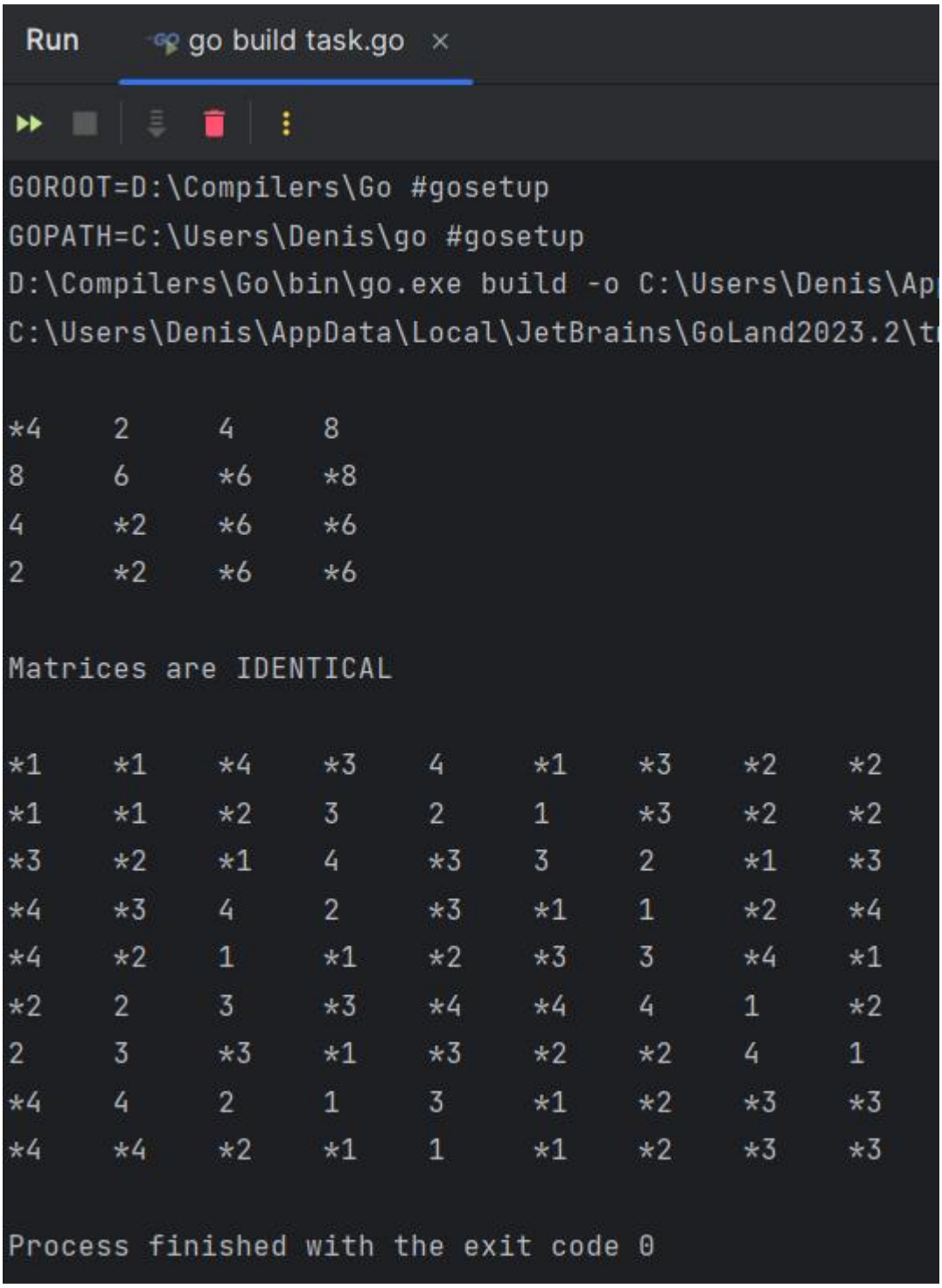
цієї клітинки виконуємо рекурсивний пошук клітини, що не задовільняє правило: “*Зафарбуйте деякі клітини так, щоб у кожному рядку або стовпці не було чисел, що повторюються.*”. За це відповідає функція checkIfUniqueWithinUnmarked, що виконує перевірку чи виконуються вимоги. Умови виходу з рекурсії: клітина знаходиться за межами матриці або клітина вже була відвідана або клітина є маркованою. Рекурсія розповсюджується по вертикалі та горизонталі.

**Результати:**




Винайдений вручну маршрут.

Примітка: числа з зірочкою - замарковані числа.


Run
go build task.go
GOROOT=D:\Compilers\Go #gosetup
GOPATH=C:\Users\Denis\go #gosetup
D:\Compilers\Go\bin\go.exe build -o C:\Users\Denis\Ap
C:\Users\Denis\AppData\Local\JetBrains\GoLand2023.2\t
*4 2 4 8
8 6 *6 *8
4 *2 *6 *6
2 *2 *6 *6
Matrices are IDENTICAL
*1 *1 *4 *3 4 *1 *3 *2 *2
*1 *1 *2 3 2 1 *3 *2 *2
*3 *2 *1 4 *3 3 2 *1 *3
*4 *3 4 2 *3 *1 1 *2 *4
*4 *2 1 *1 *2 *3 3 *4 *1
*2 2 3 *3 *4 *4 4 1 *2
2 3 *3 *1 *3 *2 *2 4 1
*4 4 2 1 3 *1 *2 *3 *3
*4 *4 *2 *1 1 *1 *2 *3 *3
Process finished with the exit code 0