Рекурсия – 2

1. Part I – Рекурсия

2. Рекурсивна сума на масив

Напишете програма, която намира сумата на всички елементи на масив от цели числа. Използвайте **рекурсия**. **Забележка**: На практика рекурсия трябва да не се използва тук (вместо това **използвайте итеративно решение**), това е просто упражнение.

Примери

Вход	Изход
1 2 3 4	10
-1 0 1	0

Подсказки

Напишете рекурсивна метод. Той ще вземе като аргументи входния масив и текущия индекс.

- Методът трябва да върнете **текущия елемент** + **сумата от всички следващите елементи** (получени чрез рекурсивно, наричайки я).
- Рекурсията трябва да спре, когато няма повече елементи в масив.

```
private static int Sum(int[] array, int index)
{
    // TODO: Set bottom of recursion
    // TODO: Return the sum of current element + sum of elements to the right
}
```

3. Рекурсивен факториел

Напишете програма, която намира факториела на дадено число. Използвайте рекурсия.

Забележка: На практика рекурсия не трябва да се използва тук (вместо това използвайте **итерация**), това е просто упражнение.

Примери

Вход	Изход
5	120
10	3628800

Подсказки

Напишете рекурсивна метод.Той взима като аргумент цяло число.

- Методът трябва да връща **текущия елемент * резултатът от изчисляване факториела на текущия елемент - 1** (получени чрез рекурсивно извикване).
- Рекурсията трябва да спре, когато няма повече елементи в масива.

```
static long Factorial(int n)
{
    // TODO: Set bottom of recursion
    // TODO: Return the multiple of current n and factorial of n - 1
}
```

4. Рекурсивно чертаене

Напишете програма, която прави фигурата по-долу в зависимост от n. Ползвайте рекурсия.

Examples

Input	Output
2	** * # ##
5	***** **** *** ** * # ## ### #### #####

Подсказки

Задаване на дъното на рекурсията

```
if (n <= 0)
{
    return;
}
```

Определете предхождащото и последващото поведение

```
Console.WriteLine(new string('*', n));
PrintFigure(n - 1);
Console.WriteLine(new string('#', n));
```

5. Генериране на 0/1 вектори

Генерирайте всички n битови вектори от нули и единици в азбучен ред.

Примери

Вход	Изход
3	000 001 010 011 100 101 110 111
5	00000 00001 00010 … 11110 11111

Подсказки

Методът трябва да получават като параметри масива, който ще бъде нашия вектор и текущ индекс

```
private static void Gen01(int[] vector, int index)
```

Дъното на рекурсията трябва да бъде, когато индексът е извън вектора

```
if (index > vector.Length - 1)
{
    // Print vector
}
```

За да генерирате всички комбинации, създайте **for** цикъл с рекурсивно извикване

```
for (int i = 0; i <= 1; i++)
{
    vector[index] = i;
    Gen01(vector, index + 1);
}
```

6. Генериране на комбинации

Генерира всички комбинации от **n** числа в групи по **k**. Прочетете множеството **от n** елементи, а след това **броят к** на елементите за избор.

Examples

Input	Output
1 2 3 4 2	1 2 1 3 1 4 2 3 2 4 3 4
1 2 3 4 5 3	1 2 3 1 2 4 1 2 5 … 3 4 5

Подсказки

Методът трябва да получат следните параметри

```
private static void GenCombs(int[] set, int[] vector, int index, int border)
{
}
```

Задаване на дъното на рекурсията

```
if (index >= vector.Length)
{
    Console.WriteLine(string.Join(" ", vector));
}
```

Преминете през всички възможни стойности за даден индекс на вектора

```
else
{
    for (int i = border; i < set.Length; i++)
    {
        vector[index] = set[i];
        GenCombs(set, vector, index + 1, i + 1);
    }
}
```

Part II – Задачата за 8 Царици

Ще реализираме рекурсивен алгоритъм за решаване на задачата за "8 царици". Нашата цел е да се напише програма за намиране на всички възможни разположения на шахматната дъска на 8 царици, така че да няма две царици, които да могат да се нападат взаимно (по ред, колона или диагонал).

Примери

Вход	Изход
(no input)	* - - - - - - - - - - - * - - - - - - - - - - * - - - - - * - - - - * - - - - - - - - - - - * - - * - - - - - - - - - * - - - - * - - - - - - - - - - - - * - - - - - - - - - * - - * - - - - - - - - - - - * - - - - * - - - - - * - - - - - - - - - - * - - - …

	(над 90 решения)

1. Научете за задачата

Научете повече за "8 царици" пъзел, например от Уикипедия:http://en.wikipedia.org/wiki/Eight_queens_puzzle.

2. Определете структурата от данни, която ще държи шахматната дъска

Първо нека да се определи структурата на данните, която ще да държи шахматната дъска. Тя трябва да се състои от 8 x 8 клетки, всяка е или заета от царица или празна. Нека също така да се определи размера на шахматната дъска като константа:

```
class EightQueens
{
    const int Size = 8;
    static bool[,] chessboard = new bool[Size, Size];
}
```

3. Определяне на структурата от данни, която държи атакуваните позиции

Ние трябва **да държим атакуваните позиции** в някаква структура от данни. Във всеки момент от изпълнението на програмата ние трябва да знаем **дали определена позиция {ред, колона} е под атака** от една царица, или не. Има много начини за съхраняване на атакуваните позиции:

- Като **пазим всички в момента поставени** царици и проверяваме дали новата позиция е в конфликт с някои от тях.
- Чрез запазване в int [,] матрица от всички атакувани позиции и проверка на новата позиция директно в него. Това ще бъде сложно да се поддържа, защото в матрицата трябва да се променят много позиции след всяка поставяне/премахване на царица.
- Като пазим множества от всички атакувани редове, колони и диагонали. Нека опитаме тази идея:

```
static HashSet<int> attackedRows = new HashSet<int>();
static HashSet<int> attackedColumns = new HashSet<int>();
static HashSet<int> attackedLeftDiagonals = new HashSet<int>();
static HashSet<int> attackedRightDiagonals = new HashSet<int>();
```

The above definitions have the following assumptions:

- **The Rows** are 8, numbered from 0 to 7.
- **The Columns** are 8, numbered from 0 to 7.
- The **left diagonals** are 15, numbered from -7 to 7. We can use the following formula to calculate the left diagonal number by row and column: **`leftDiag = col - row`**.
- The **right diagonals** are 15, numbered from 0 to 14 by the formula: **`rightDiag = col + row`**.

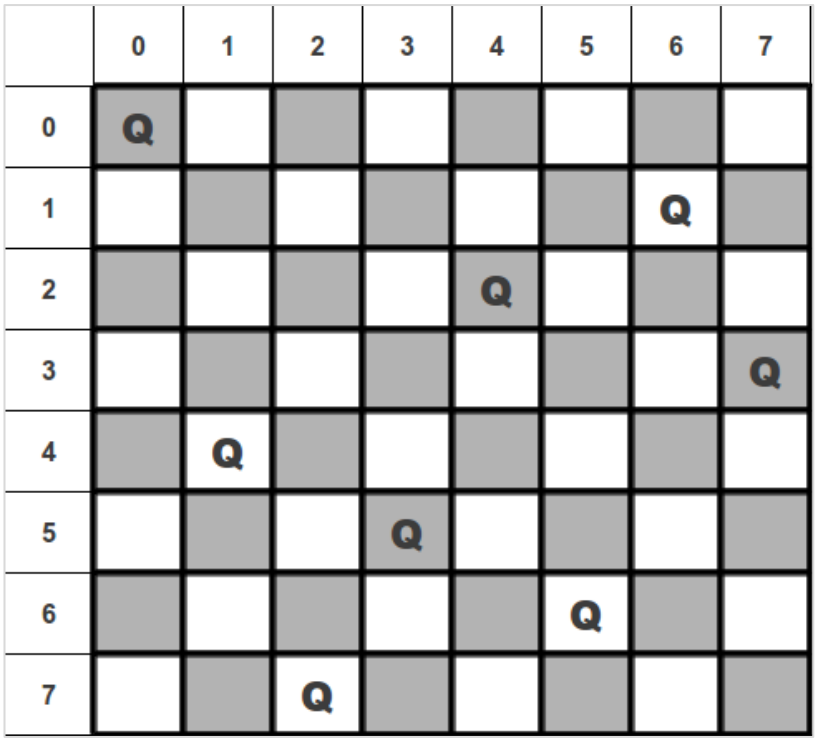
Let's take as an **example** the following chessboard with 8 queens placed on it at the following positions:

- {0, 0}; {1, 6}; {2, 4}; {3, 7}; {4, 1}; {5, 3}; {6, 5}; {7, 2}

Горните определения имат следните предположения:

- редове са 8, номерирани от 0 до 7.
- Колоните са 8, номерирани от 0 до 7.
- левите диагонали са 15, номерирани от -7 до 7. Можем да използваме следната формула за изчисляване на номера на левия диагонал по ред и колона: **leftDiag = col - row**.
- десните диагонали са 15, номерирани от 0 до 14 по формулата: rightDiag = col + ред.

Да вземем като **пример** следната шахматната дъска с 8 царици поставени върху ѝ в следните позиции: {0, 0}; {1, 6}; {2, 4}; {3, 7}; {4, 1}; {5, 3}; {6, 5}; {7, 2}



Следвайки определенията по-горе за нашия пример царица {4, 1} заема ред 4, колона 1, левия диагонал -3 и десния диагонал 5.

4. Напишете Backtracking алгоритъм

Сега е време да се напише на рекурсивен backtraking алгоритъм за разполагането на 8 царици. Алгоритъм започва от ред 0 и се опитва да постави кралица в някоя колона в ред 0. При успех той се опитва да постави следващата царица в ред 1, после следващата царица на ред 2 и т.н. до последния ред. Кодът за поставянето на следващата царица в определен ред може да изглежда така:

```
static void PutQueens(int row)
{
    if (row == Size)
    {
        PrintSolution();
    }
    else
    {
        for (int col = 0; col < Size; col++)
        {
            if (CanPlaceQueen(row, col))
            {
                MarkAllAttackedPositions(row, col);
                PutQueens(row + 1);
                UnmarkAllAttackedPositions(row, col);
            }
        }
    }
}
```

Първоначално ние извикваме този метод от ред 0:

```
static void Main()
{
    PutQueens(0);
}
```

5. Проверка, ако позицията е свободна

Now, let's write **the code to check whether a certain position is free**. A position is free when it is not under attack by any other queen. This means that if some of the rows, columns or diagonals is already occupied by another queen, the position is occupied. Otherwise it is free. A sample code might look like this: Сега нека да напишем код, за да проверим дали е свободна определена позиция. Позицията е свободна, когато не е под атака от всички други царици. Това означава, че ако някои редове, колони и диагонали вече са заети от други царици, то позицията е заета. В противен случай тя е свободна. Примерно, кода може да изглежда така:

```
static bool CanPlaceQueen(int row, int col)
{
    var positionOccupied =
         attackedRows.Contains(row) ||
         attackedColumns.Contains(col) ||
         attackedLeftDiagonals.Contains(col - row) ||
         attackedRightDiagonals.Contains(row + col);
    return !positionOccupied;
}
```

Recall that **`col-row`** is the number of the left diagonal and **`row+col`** is the number of the right diagonal. Спомнете си, че **`col-row`** е номера на левия диагонал и че **`row+col`** е номера на десния диагонал.

6. Маркиране / Демаркиране на полетата, които царицата атакува

After a queen is placed, we need to **mark as occupied all rows, columns and diagonals** that it can attack: След като царицата е поставена, ние трябва да маркирате като заети всички редове, колони и диагонали, които тя може да атакува

```
static void MarkAllAttackedPositions(int row, int col)
{
    attackedRows.Add(row);
    attackedColumns.Add(col);
    attackedLeftDiagonals.Add(col - row);
    attackedRightDiagonals.Add(row + col);
    chessboard[row, col] = true;
}
```

On removal of a queen, we will need a method to mark as free all rows, columns and diagonals that were attacked by it. Write it yourself: При преместване на царица ще ни трябва метод, който да маркира като свободни всички редове, колони и диагонали, които са били нападнати от него. Напишете сами:

```
static void UnmarkAllAttackedPositions(int row, int col)
{
    // TODO
}
```

7. Отпечатайте решението

Когато е намерено решение, то трябва да бъде отпечатано в конзолата. Първо въведе брояч на решения да се опрости проверка дали намерените решения са правилни:

```
class EightQueens
{
    static int solutionsFound = 0;
```

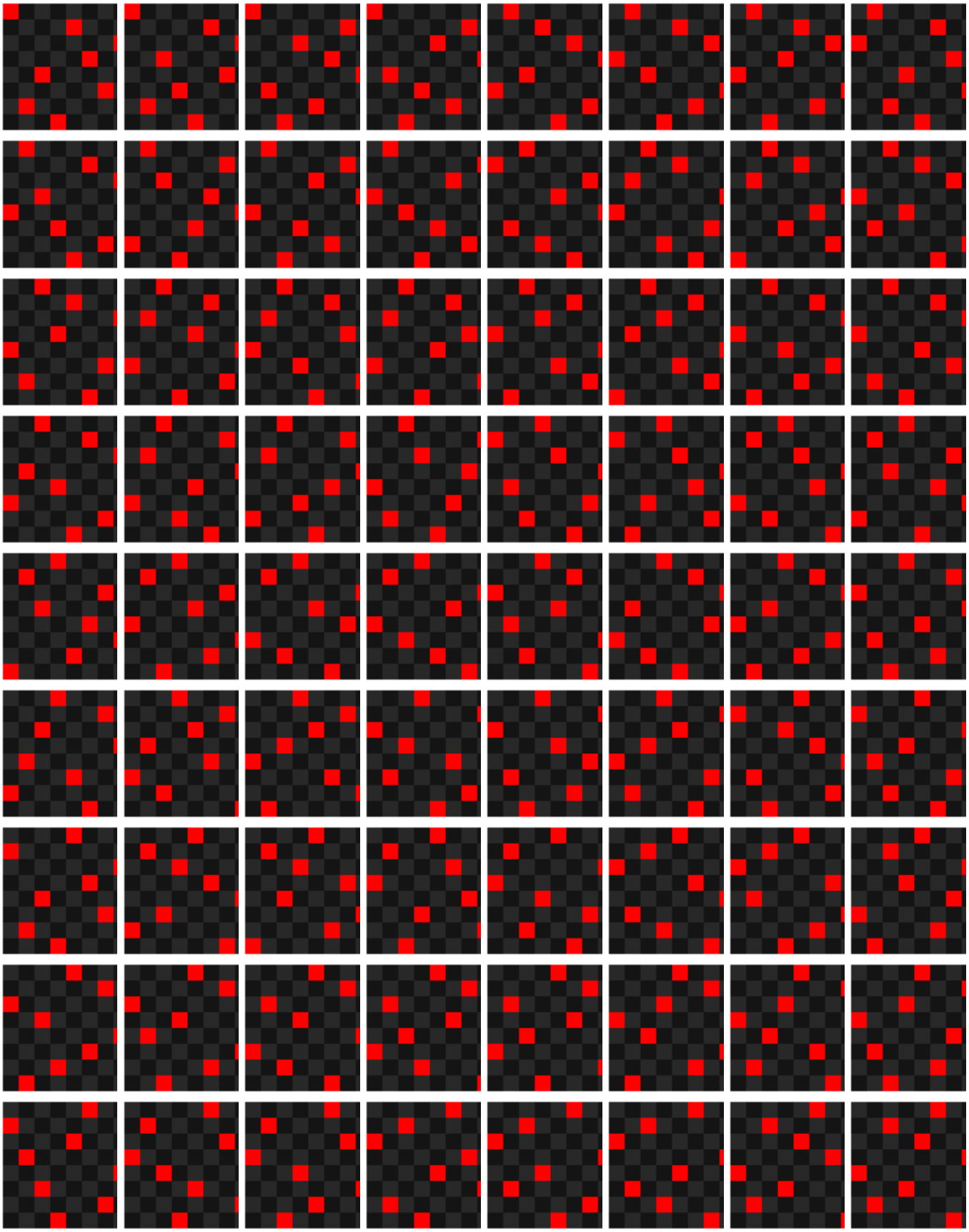
След това преминете през всички редове и всички колони на шахматната дъска и ги отпечатайте:

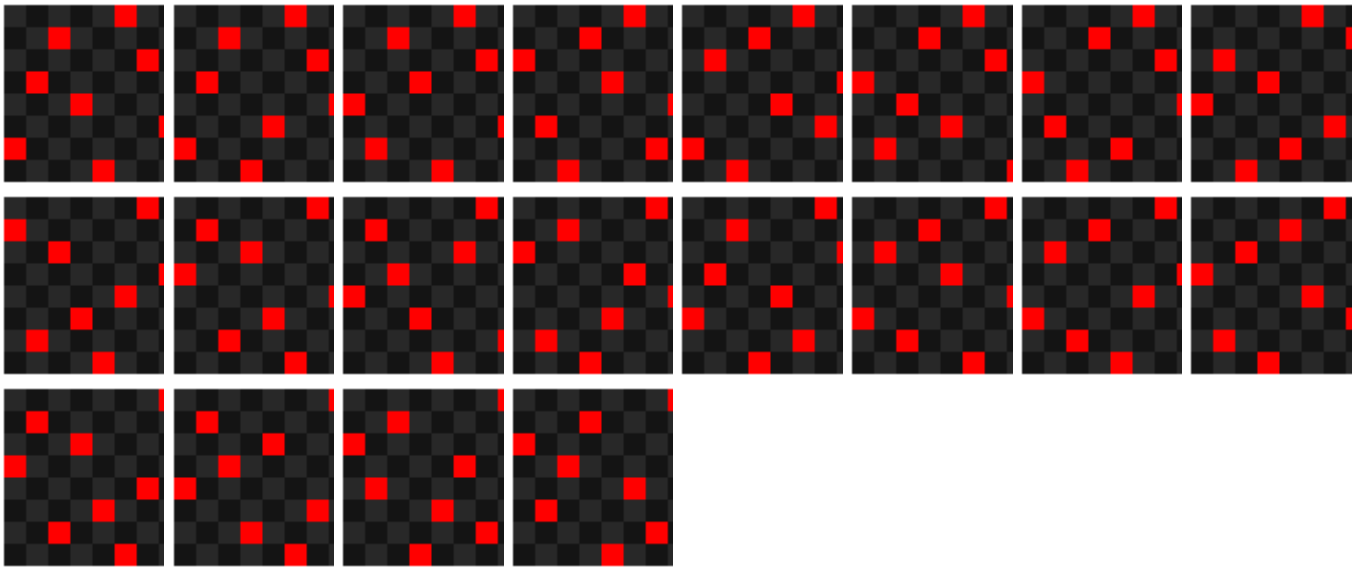
```
static void PrintSolution()
{
    for (int row = 0; row < Size; row++)
    {
        for (int col = 0; col < Size; col++)
        {
            // TODO: print '*' or '-' depending on scoreboard[row, col]
        }
        Console.WriteLine();
    }
    Console.WriteLine();

    solutionsFound++;
}
```

8. Testing the Code

Задачата за "8 царици" има 92 **различни решения**. Проверете дали вашия код генерира и отпечатва всички тях правилно. **SolutionsFound** брояч ще ви помогне да проверите броя на решения. По-долу са 92 отделни решения:





9. Оптимизирайте решението

Сега можем да оптимизираме нашия код:

- премахване на множеството **attackedRows**. Не е необходимо, защото всички кралици са поставени последователно в редове 0... 7. Опитайте се да използвате **bool** [] масив за **attackedColumns**, **attackedLeftDiagonals** и **attackedRightDiagonals** вместо множества. Обърнете внимание, че масивите са индексирани от 0 до техния размер и не може да има отрицателни индекси.

* Решение с пермутации

Try to implement the more-efficient **permutation-based solution** of the "8 Queens" puzzle. Look at this code to grasp the ide Опитайте се да приложите по-ефективни пермутации, както в задачата "8 царици". Погледнете този код да се възползвате от идеята http://introcs.cs.princeton.edu/java/23recursion/Queens.java.html.

Part III – Намиране на път в лабиринт

Даден е лабиринт. Целта ви е да намерите всички пътища от самото начало (клетка 0, 0) към изхода, маркирана с 'e'.

- Празните клетки са отбелязани с тире "-".
- Стените са отбелязани със звездичка "*".

На първия ред вие ще получите размерите на лабиринта. На следващите ще получите самия лабиринт. Редът на пътища не е от значение.

Примери

Input	Output
3 3 --- -*-	RRDD DDRR

--e	
3 5 -**-e ----- *****	DRRRRU DRRRUR

Подсказка

Създаване на методи за четене и намирането на всички пътища в лабиринта.

```
static void Main(string[] args)
{
    lab = ReadLab();
    FindPaths(0, 0, 'S');
}
```

Създаване на статичен списък, който ще съдържа всички пътища

```
static List<char> path = new List<char>();
```

Намиране на всички пътища трябва да е рекурсивно

```
private static void FindPaths(int row, int col, char direction)
{
    if (!IsInBounds(row, col))
        return;

    path.Add(direction);

    if (IsExit(row, col))
    {
        PrintPath();
    }
    else if (!IsVisited(row, col) && IsFree(row, col))
    {
        Mark(row, col);
        FindPaths(row, col + 1, 'R');
        FindPaths(row + 1, col, 'D');
        FindPaths(row, col - 1, 'L');
        FindPaths(row - 1, col, 'U');
        Unmark(row, col);
    }

    path.RemoveAt(path.Count - 1);
}
```

Изпълнете всички helper методи, които се намират в горния код.

Министерство на образованието и науката (МОН)

- Настоящият курс (презентации, примери, задачи, упражнения и др.) е разработен за нуждите на Национална програма "**Обучение за ИТ кариера**" на МОН за подготовка по професия "Приложен програмист".

- Курсът е базиран на учебно съдържание и методика, предоставени от **фондация "Софтуерен университет"** и се разпространява под **свободен лиценз CC-BY-NC-SA** (Creative Commons Attribution-Non-Commercial-Share-Alike 4.0 International).