# Решения на задачите от глава 10 на книгата "Въведение в програмирането със C#"

Предлагаме ви решения на задачите от [книгата "Въведение в програ­мирането със C#"](http://www.introprogramming.info), заедно с анализ на задачата, описание на използва­ните идеи, алгоритми, подходи за решаване и тестове.

## Авторски колектив

**Борис Красимиров Гуцев**

Контакти: (bgutsev@gmail.com)

**Волен Вълков**

Контакти: (volen.vulkov@gmail.com)

**Ивайло Пламенов Гергов**

Контакти: (ivaylo.gergov@gmail.com)

**Симеон Димитров**

Контакти: ([smndimitrov@gmail.com](mailto:smndimitrov@gmail.com))

|  |  |
| --- | --- |
| Задача 10 – Път между две клетки в лабиринт | |
| **Условие**  Дадена е матрица с проходими и непроходими клетки. Напишете рекурсивна програма, която намира всички пътища между две клетки в матрицата. | |
| **Описание на входа**   * На първият ред въвеждаме размера на матрицата – броят редове последван от интервал и броят колони. * След това веждаме матрицата, като за проходима клетка използваме ‚-‘, а за непроходима ‚\*‘. * Въвеждаме индекса на входящата клетка, първо номерът на реда последван от интервал и номерът на колоната. * Въвеждаме индекса на изходящата клетка първо номерът на реда последван от интервал и номерът на колоната. | |
| **Описание на изхода**  Всеки намерен път се изкарва на нов. Пътят е във следният формат, като всяка буква от реда е посоката на стъпката при търсенето на път между двете клетки:   * R – right * L – left * U – up * D – down   Ако програмата не намери изход на екрана се отпечатва “NO EXIT FOUND!” | |
| **Анализ на задачата**  Потребителят въвежда първо размера на матрицата, а след това въвежда матрицата ред по ред. В следващите два реда потребителят въвежда индексите на входните и изходните клетки, разделени с интервал.  Лабиринта се съхранява като статичен двумерен char array, за да не го предаваме всеки път като аргумент на рекрусивната функция, която ще намира пътя.  За да можем да намерим пътя от една клетка до друга ще ни трябва някакъв масив който да пази стъпките които сме направили, тоест във каква посока сме се движили. Този масив ще съдържа във всеки един момент текущия път от началото на лабиринта до текущата позиция. В случея ще използваме StringBuilder за съхраняване на пътя.  Също така ще декларираме булева променлива findExit, и ще я инициализираме с false. Тази променлива ще бъде false, докато не бъде намерен път.  Добавихме още един параметър на функцията за търсене на път между две клетки: посоката, в която сме поели, за да стигнем текущата позиция. Този параметър няма смисъл при първо­на­чалното започване от стартовата позиция и затова в началото слагаме за посока някаква безсмислена стойност ‘**b**‘. След това при отпечатването пропускаме първия елемент от пътя.  Преди да влезем в рекурсивният метод променяме стойноста на клетката, която потребителят е въвел като изходна, а на рекурсивният метод, подаваме като параметър индекса на клетката (съответно ред и колона), от която ще търсим път.  При влизане в рекурсивния метод за търсене първо се проверява дали няма излизане извън лабиринта. Ако има, търсенето от текущата позиция нататък се прекратява, защото е забранено излизане извън границите на лабиринта.  След това се проверява дали не сме намерили изхода. Ако сме го намерили, се отпечатва масива който пази направените стъпки, променяме стойноста на булевата променлива findExit и търсенето от текущата позиция нататък приключва.  След това се проверява дали е свободна текущата клетка. Клетката е свободна, ако е проходима и не сме били на нея при някоя от предните стъпки (ако не е част от текущия път от стартовата позиция до текущата клетка на лабиринта).  При свободна клетка, се осъществява стъпване в нея. Това се извършва като се означи клет­ката като заета (със символа ‘**v**‘). След това рекур­сивно се търси път в четирите възможни посоки. След връщане от рекур­сивното проучване на четирите възможни посоки, се отстъпва назад от текущата клетка и тя се маркира отново като свободна (връщане назад).  Рекрусивният метод e void, и това ни олеснява, защото ако не сме на изхода или на свободна клетка метода приключва без връщана стойност или се връща в миналият си етап от рекурсията.  Маркирането на текущата клетка като свободна при излизане от рекурси­ята е важно, защото при връщане назад тя вече не е част от текущия път. Ако бъде пропуснато това действие, няма да бъдат намерени всички пътища до изхода, а само някои от тях.  Програмата приключва когато се обходят всички възможни клетки с така нареченият backtracking. Ако не бъде намерен път се отпечатва подходящо съобщение. | |
| **Решение (сорс код)** | |
| using System;  using System.Linq;  using System.Text;  namespace LabPaths  {  class LabyrinthPaths  {  static char[,] lab;  static void MarkExit(int row, int col)  {  lab[row, col] = 'e';  }    static bool findExit = false;  static StringBuilder path = new StringBuilder { };  static void FindPaths(char direction, int col, int row)  {  if ( col >= lab.GetLength(1) || row>= lab.GetLength(0) || row < 0 || col < 0 )  {  return;  }  path.Append(direction);  if (lab[row, col] == 'e')  {  findExit = true;  PrintPath(path);  }  if (lab[row, col] == '-')  {  lab[row, col] = 'v'; //mark as visited  FindPaths('L', col - 1, row);  FindPaths('R', col + 1, row);  FindPaths('U', col, row - 1);  FindPaths('D', col, row + 1);  lab[row, col] = '-';  }  path.Remove(path.Length - 1, 1);  }  static void PrintPath(StringBuilder path)  {  for (int i = 1; i < path.Length; i++)  {  Console.Write(path[i]);  }  Console.WriteLine();  }  static void Main(string[] args)  {  string labyrinthSize = Console.ReadLine();  string[] labSize = labyrinthSize.Split(' ');  lab = new char[int.Parse(labSize[0]), int.Parse(labSize[1])];  string row;  for (int i = 0; i < lab.GetLength(0); i++)  {  row = Console.ReadLine();  for (int j = 0; j < row.Length; j++)  {  lab[i, j] = row[j];  }  }  string entranceCell;  entranceCell = Console.ReadLine();  string[] eCellCordinates = entranceCell.Split(' ');  string exitCell;  exitCell = Console.ReadLine();  string[] exitCellCordinates = exitCell.Split(' ');  if (eCellCordinates[0] == exitCellCordinates[0] && eCellCordinates[1] == exitCellCordinates[1])  {  return;  }  MarkExit(int.Parse(exitCellCordinates[0]), int.Parse(exitCellCordinates[1]));  FindPaths('b', int.Parse(eCellCordinates[0]), int.Parse(eCellCordinates[1]));  if (findExit == false)  {  Console.WriteLine("NO EXIT FOUND!");  }  }  }  } | |
| **Тестове**  Програмата е тествана:   * с лабиринт само от проходими клетки и с лабиринт само от непроходими. * При подаване на едни и същи стойности за входна и изходна клетка * При лабиринт само от една клетка (в тези два случея има път който е без нито една стъпка и прогарамата не отпечатва нищо) * При по голям лабиринт (20х20) с много възможни пътища * При лабиринт без възможен път | |
| **Вход** | **Изход** |
| 2 2  --  --  0 0  1 1 | RD  DR |
| **Вход** | **Изход** |
| 3 3  -\*-  \*\*-  --\*  1 2  2 2 | NO EXIT FOUND! |

# Глава 10. Рекурсия

1. Напишете програма, която симулира изпълнението на n вложени цикъла от 1 до n. Пример:

1 1 1

1 1 2

1 1 3

1 1 1 2 1

n=2 -> 1 2 n=3 -> ….

2 1 3 2 3

2 2 3 3 1

3 3 2

3 3 3

1. Напишете рекурсивна програма, която генерира и отпечатва всички комбинации с повторение на k елемента над n-елементно множество.

Примерен вход:

|  |
| --- |
| n = 3  k = 2 |

Примерен изход:

|  |
| --- |
| (1 1), (1 2), (1 3), (2 2), (2 3), (3 3) |

Измислете и реализирайте итеративен алгоритъм за същата задача.

1. Напишете рекурсивна програма, която генерира всички вариации с повторение на n елемента от k-ти клас.

Примерен вход:

|  |
| --- |
| n = 3  к = 2 |

Примерен изход:

|  |
| --- |
| (1 1), (1 2), (1 3), (2 1), (2 2), (2 3), (3 1), (3 2), (3 3) |

Измислете и реализирайте итеративен алгоритъм за същата задача.

1. Нека е дадено множество от символни низове. Да се напише рекур­сивна програма, която генерира всички подмножества съставени от точно k на брой символни низа, избрани измежду елементите на това множество.

Примерен вход:

|  |
| --- |
| strings = {'test', 'rock', 'fun'}  k = 2 |

Примерен изход:

|  |
| --- |
| (test rock), (test fun), (rock fun) |

Измислете и реализирайте итеративен алгоритъм за същата задача.

1. Напишете рекурсивна програма, която отпечатва всички подмножества на дадено множество от думи.

Примерен вход:

|  |
| --- |
| words = {'test', 'rock', 'fun'} |

Примерен изход:

|  |
| --- |
| (), (test), (rock), (fun), (test rock), (test fun), (rock fun), (test rock fun) |

Измислете и реализирайте итеративен алгоритъм за същата задача.

1. Реализирайте алгоритъма "сортиране чрез сливане" (merge-sort). При него началният масив се разделя на две равни по големина части, които се сортират (рекурсивно чрез merge-sort) и след това двете сортирани части се сливат, за да се получи целият масив в сортиран вид.
2. Напишете рекурсивна програма, която генерира и отпечатва пермута­циите на числата 1, 2, …, n, за дадено цяло число n.

Примерен вход:

|  |
| --- |
| n = 3 |

Примерен изход:

|  |
| --- |
| (1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1) |

1. Даден е масив с цели числа и число N. Напишете рекурсивна прог­рама, която намира всички подмножества от числа от масива, които имат сума N. Например ако имаме масива {2, 3, 1, -1} и N=4, можем да получим N=4 като сума по следните два начина: 4=2+3-1; 4=3+1.
2. Даден е масив с цели **положителни** числа. Напишете програма, която проверява дали в масива съществуват едно или повече числа, чиято сума е N. Можете ли да решите задачата без рекурсия?
3. Дадена е матрица с проходими и непроходими клетки. Напишете рекурсивна програма, която намира всички пътища между две клетки в матрицата.
4. Модифицирайте горната програма, за да проверява дали съществува път между две клетки без да се намират всички възможни пътища. Тествайте за матрица 100х100 пълна само с проходими клетки.
5. Напишете програма, която намира най-дългата поредица от съседни проходими клетки в матрица.
6. Даден е двумерен масив с проходими и непроходими клетки. Напишете програма, която намира всички площи съставени само от проходими клетки.
7. Реализирайте алгоритъма BFS (breath-first search) за търсене на най-кратък път в лаби­ринт. Ако се затруднявате, потърсете информация в Интернет.
8. Напишете рекурсивна програма, която обхожда целия твърд диск C:\ рекурсивно и отпечатва всички папки и файловете в тях.