# 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

**Виктория Димитрова Димитрова**

e-mail: viktoriyadd@gmail.com

**Vladimir Ivanov Iliev**

e-mail: vladimirivanoviliev@gmail.com

**Vladimir Hristov Tsenev**

e-mail: vladimir.tsenev@gmail.com

**Франц Фишбах**

e-mail: FranzFischbach@gmail.com

skype: Ugabuk

# Решения на задачите

|  |  |
| --- | --- |
| Задача 1. N- вложени цикъла | |
| **Условие**   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 | |
| **Описание на входа**  Входът е едно чило – броя на вложените цикъла, които трябва да симулира програмата. | |
| **Описание на изхода**  Всяка комбинация (генерирана от най-вложения цикъл) е на нов ред и след всяко число на даден ред има по един празен символ. | |
| **Анализ на задачата**  Резултатът представлява всички възможни комбинации от n числа. Има nn комбинации. | |
| **Решение (сорс код)**  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace SimulateNLoops  {  class Program  {  static void PrintNumbers(int[] n, int iter)  {  if (iter>=n.Length)  {  foreach (int element in n)  {  Console.Write(element + " ");  }  Console.WriteLine();  }  else  {  for (int i = 1; i <= n.Length; i++)  {  n[iter] = i;  PrintNumbers(n, iter + 1);  }  }  }  static void Main(string[] args)  {  int n = int.Parse(Console.ReadLine());  if (numberOfLoops < 1)  {  throw new ArgumentException("Number of loops should be 1 or higher.");  }  int[] arr = new int[n];  PrintNumbers(arr, 0);  }  }  } | |
|  | |
| **Тестове** | |
| **Вход** | **Изход** |
| 2 | 1 1  1 2  2 1  2 2 |
| **Вход** | **Изход** |
| 3 | 1 1 1  1 1 2  1 1 3  1 2 1  1 2 2  1 2 3  1 3 1  1 3 2  1 3 3  2 1 1  2 1 2  2 1 3  2 2 1  2 2 2  2 2 3  2 3 1  2 3 2  2 3 3  3 1 1  3 1 2  3 1 3  3 2 1  3 2 2  3 2 3  3 3 1  3 3 2  3 3 3 |
| **Вход** | **Изход** |
| 4 | 1 1 1 1  1 1 1 2  . . . .  4 4 4 3  4 4 4 4 |
| **Вход** | **Изход** |
| 1 | 1 |
| **Вход** | **Изход** |
| N >= 7 | Изпълнението на този тест изисква много време и е за да добиете реална представа за това колко време е необходимо на компютъра ви да обходи дадено N-мерно пространство. |
| **Вход** | **Изход** |
| N <= 0 | Съобщение за грешен вход. |

|  |  |
| --- | --- |
| Задача 2. Комбинации с повторения | |
| **Условие**  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)** | | |
| **Описание на входа**  На първия ред прочитаме числото n, на втория – k. | |
| **Описание на изхода**  Различните комбинации от k-ти клас над n елемента се изреждат, като всяка комбинация е на нов ред и след всяко число има един празен символ. | |
| **Анализ на задачата**  Здачата е реализирана върху тази за N- вложени циклъла, само че се избягват повторенията и всеки пък при извикването на рекурсията се премахват вече миналите компоненти. | |
| **Решение (сорс код)**  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace Combinations  {  class Program  {  static int[] combinationsArray;  static void Main(string[] args)  {  int maxNumber = 0;  maxNumber = int.Parse(Console.ReadLine());  int maxPosition = 0;  maxPosition = int.Parse(Console.ReadLine());  combinationsArray = new int[maxPosition];  SimulateNestedLoops(maxNumber, maxPosition);  }  static void SimulateNestedLoops(int maxNumber, int maxPosition, int startNumber = 1, int currentPosition = 1)  {  for (int i = startNumber; i <= maxNumber; i++)  {  combinationsArray[currentPosition - 1] = i;  if (currentPosition != maxPosition)  {  SimulateNestedLoops(maxNumber, maxPosition, startNumber, currentPosition + 1);  startNumber++;  }  else  {  PrintArray(combinationsArray);  }  }  }  static void PrintArray(int[] numbersArray)  {  for (int i = 0; i < numbersArray.Length; i++)  {  Console.Write("{0} ", numbersArray[i]);  }  Console.WriteLine();  }  }  } | |
|  | |
| **Тестове** | |
| **Вход** | **Изход** |
| 3  2 | 1 1  1 2  1 3  2 2  2 3  3 3 |
| **Вход** | **Изход** |
| 4  3 | 1 1 1  1 1 2  1 1 3  1 1 4  1 2 2  1 2 3  1 2 4  1 3 3  1 3 4  1 4 4  2 2 2  2 2 3  2 2 4  2 3 3  2 3 4  2 4 4  3 3 3  3 3 4  3 4 4  4 4 4 |

|  |  |
| --- | --- |
| Задача 3. Вариации с повторения | |
| **Условие**  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)** | | |
| **Описание на входа**  На първия ред прочитаме числото n, на втория – k. | |
| **Описание на изхода**  Различните вариации от k-ти клас над n елемента се изреждат, като всяка вариация е на нов ред и след всяко число има един празен символ. | |
| **Анализ на задачата**  Трябва да се генерират всички множества от **k** елемента в диапазона **[0...N-1]**. За всяко такова множество се разглеждат числата от него като индекси в масива и се отпечатва за всяко число. | |
| **Решение (сорс код)**  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace VariationWithDuplicates  {  class Program  {  static void Variations(int[] output, int n, int iter, int start)  {  if (iter >= output.Length)  {  foreach (int value in output)  {  Console.Write(valuе + " ");  }  Console.WriteLine();  }  else  {  for (int i = 1; i <= n; i++)  {  output[iter] = i;  Variations(output, n, iter + 1, i);  }  }  }  static void Main(string[] args)  {  int n = int.Parse(Console.ReadLine());  int k = int.Parse(Console.ReadLine());  int[] arr = new int[k];  Variations(arr, n, 0, 1);  }  }  } | |
|  | |
| **Тестове** | |
| **Вход** | **Изход** |
| 3  2 | 1 1  1 2  1 3  2 1  2 2  2 3  3 1  3 2  3 3 |
| **Вход** | **Изход** |
| 4  2 | 1 1  1 2  1 3  1 4  2 1  2 2  2 3  2 4  3 1  3 2  3 3  3 4  4 1  4 2  4 3  4 4 |

|  |  |
| --- | --- |
| Задача 4. Подмножества от К елемента | |
| **Условие**  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)** | | |
| **Описание на входа**  На първия ред на входа е числото n = броя на символни низове. На следващите n реда се въвежат низовете, като всеки е на нов ред. Последната стъпка е да се въведе числото k. | |
| **Описание на изхода**  Възможните подмножества са изведени всяко на нов ред с празен символ между тях. | |
| **Анализ на задачата**  Най-подходящото решение е свързано с генерираето на вектор в двуична форма. 001 – 010 – 011 и т.н. като К е най-холямото число. От всички възможни варианти до К се транформират двучните числа в дадените символни низове, чрез позицията им. | |
| **Решение (сорс код)**  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace SubsetOfStrings  {  class Program  {  static string[] setOfStrings;  static void Main(string[] args)  {  int length = int.Parse(Console.ReadLine());  setOfStrings = new string[length];  for (int i = 0; i < length; i++)  {  setOfStrings[i] = Console.ReadLine();  }  int k = int.Parse(Console.ReadLine());  int[] array = new int[k];  FindCombinations(array, 0, 0, length);  }  static void FindCombinations(int[] vector, int index, int start, int end)  {  if (index >= vector.Length)  {  Print(vector);  }  else  {  for (int i = start; i < end; i++)  {  vector[index] = i;  FindCombinations(vector, index + 1, i + 1, end);  }  }  }  static void Print(int[] vector)  {  for (int i = 0; i < vector.Length; i++)  {  Console.Write("{0}", setOfStrings[vector[i]]);  if (i != vector.Length - 1)  {  Console.Write(" ");  }  }  Console.WriteLine();  }  }  } | |
|  | |
| **Тестове** | |
| **Вход** | **Изход** |
| 3  test  rock  fun  2 | test rock  test fun  rock fun |
| **Вход** | **Изход** |
| 4  test  rock  fun  .net  3 | test rock fun  test rock .net  test fun .net  rock fun .net |

|  |  |
| --- | --- |
| Задача 5. Подмножества от думи | |
| **Условие**  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)** | | |
| **Описание на входа**  На първия ред на входа е числото n = броя на символни низове. На следващите n реда се въвежат низовете, като всеки е на нов ред. | |
| **Описание на изхода**  Всички подмножества на думите в множеството се отпечатват на един ред, като всяко подмножество е оградено в скоби и след всяко освен последното има запетая и празен символ. | |
| **Анализ на задачата**  Броят на подмножествата е равен на 2n, където n е броят на думите в множеството (в подмножествата се включва и празното множество). | |
| **Решение (сорс код)**  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace Recursion\_Task5  {  class Program  {  private static List<string> wordSequence = new List<string>(){ "test", "rock", "fun" };  private static StringBuilder output = new StringBuilder();  private static List<string> allSubsequences = new List<string>();  static void Main(string[] args)  {  int length = int.Parse(Console.ReadLine());  wordSequence = new List<string>();  for (int i = 0; i < length; i++)  {  string word = Console.ReadLine();  wordSequence.Add(word);  }  SolveWithRecursion(0, "");  Console.WriteLine(output.ToString().TrimEnd(',', ' '));  }  private static void SolveWithRecursion(int index, string set)  {  if (allSubsequences.Count == 1 << wordSequence.Count)  {  return;  }  if (index == wordSequence.Count)  {  if (!allSubsequences.Contains(set))  {  allSubsequences.Add(set);  string result = "(" + set.TrimEnd() + "), ";  output.Append(result);  }  set = "";  return;  }  for (int i = 0; i < wordSequence.Count; i++)  {  SolveWithRecursion(index + 1, set);  if (!set.Contains(wordSequence[index]))  {  set += wordSequence[index] + " ";  }  }  }  }  } | |
|  | |
| **Тестове** | |
| **Вход** | **Изход** |
| 3  test  rock  fun | (), (fun), (rock), (rock fun), (test), (test fun), (test rock), (test rock fun) |
| **Вход** | **Изход** |
| 4  test  rock  fun  .net | (), (.net), (fun), (fun .net), (rock), (rock .net), (rock fun), (rock fun .net), (test), (test .net), (test fun), (test fun .net), (test rock), (test rock .net), (test rock fun), (test rock fun .net) |

|  |  |
| --- | --- |
| Задача 6. Сортиране чраз сливане | |
| **Условие**  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. | |
| **Описание на входа**  На първия ред е число n от целочислен тип равно на броя елементи, които трябва да бъдат сортирани. На следващите n реда се въвеждат елементите. | |
| **Описание на изхода**  Изходът е на един ред и ограден от къдрави скоби. Между скобите са изписани всички числа в сортиран вид, като между всеки две има запетая и празен символ. | |
| **Анализ на задачата**  Даденият арлгоритъм е описан навсякъде. В сегашната реализация се използва List<>, за по-бързо добавяне и изтриване на елементите. | |
| **Решение (сорс код)**  using System;  using System.Collections.Generic;  using System.Linq;  namespace MergeSort  {  class Program  {  static void Main(string[] args)  {  int listCount = int.Parse(Console.ReadLine());  List<int> list = new List<int>();  for (int i = 0; i < listCount; i++)  {  int number = int.Parse(Console.ReadLine());  list.Add(number);  }  List<int> sortedList = MergeSort(list);  PrintList(sortedList);  }  static void PrintList(List<int> list)  {  Console.Write("{ ");  for (int i = 0; i < list.Count - 1; i++)  {  Console.Write("{0}, ", list[i]);  }  Console.Write("{0} ", list[list.Count - 1]);  Console.WriteLine("}");  }  static List<int> MergeSort(List<int> arr)  {  if (arr.Count <= 1)  {  return arr;  }  List<int> left = new List<int>();  List<int> right = new List<int>();  List<int> result = new List<int>();  int middle = arr.Count / 2;  for (int i = 0; i < arr.Count; i++)  {  if (i < middle)  {  left.Add(arr[i]);  }  else  {  right.Add(arr[i]);  }  }  left = MergeSort(left);  right = MergeSort(right);  result = Merge(left, right);  return result;  }  static List<int> Merge(List<int> left, List<int> right)  {  List<int> result = new List<int>();  while (left.Count > 0 || right.Count > 0)  {  if (left.Count > 0 && right.Count > 0)  {  if (left[0] <= right[0])  {  result.Add(left[0]);  left.RemoveAt(0);  }  else  {  result.Add(right[0]);  right.RemoveAt(0);  }  }  else if (left.Count > 0)  {  result.Add(left[0]);  left.RemoveAt(0);  }  else if (right.Count > 0)  {  result.Add(right[0]);  right.RemoveAt(0);  }  }  return result;  }  }  } | |
|  | |
| **Тестове** | |
| **Вход** | **Изход** |
| 5  5  4  3  2  1 | { 1, 2, 3, 4, 5 } |
| **Вход** | **Изход** |
| 3  1  3  2 | { 1, 2, 3 } |

|  |  |
| --- | --- |
| Задача 7. Пермутации | |
| **Условие**   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)** | | |
| **Описание на входа**  Входът е на един ред. Въвежда се числото n. | |
| **Описание на изхода**  Всяка пермутация е на нов ред. Числата в пермутацията се изписват едно след друго без празни символи. | |
| **Анализ на задачата** | |
| **Решение (сорс код)**  using System;  namespace Permutations  {  class Program  {  static void GenPermutations(int[] arr, int index)  {  if (index == arr.Length)  {  foreach (int value in arr)  {  Console.Write(value);  }  Console.WriteLine();  }  else  {  for (int i = index; i < arr.Length; i++)  {  int temp = arr[i];  arr[i] = arr[index];  arr[index] = temp;  GenPermutations(arr, index + 1);  temp = arr[i];  arr[i] = arr[index];  arr[index] = temp;  }  }  }  static void Main(string[] args)  {  int n = int.Parse(Console.ReadLine());  int[] array = new int[n];  for (int i = 0; i < array.Length; i++)  {  array[i] = i + 1;  }  GenPermutations(array, 0);  }  }  } | |
|  | |
| **Тестове** | |
| **Вход** | **Изход** |
| 3 | 123  132  213  231  321  312 |
| **Вход** | **Изход** |
| 2 | 12  21 |

|  |  |
| --- | --- |
| Задача 8. Подмножества от числа със сума N | |
| **Условие**  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**. | |
| **Описание на входа**  На първия ред се въвежда броя на елементи в масива n. На следващите n реда се въвеждат елементите му. На последния ред на входа се въвежда сумата, която искаме да получим. | |
| **Описание на изхода**  Всички подмножества с дадената сума се изписват оградени от къдрави скоби и с празни символи между числата в подмножеството. Всяко подмножество е на нов ред. | |
| **Анализ на задачата**  Трябва да се търсят всички възможни комбинации, така че сумата да е равна на N. Задачата няма ефективно решение, особено при по-голямо число и повече добре подбрани числа в масива. | |
| **Решение (сорс код)**  using System;  using System.Collections.Generic;  using System.Linq;  namespace Recursion\_Task8  {  class Program  {  private static int k;  private static int arraySize;  private static int[] used;  private static int[] combo;  private static List<int[]> combinations;  private static int sum;  private static int[] numbers;  static void Main(string[] args)  {  ProcessInput();  arraySize = numbers.Length;  used = new int[arraySize];  combinations = new List<int[]>();  for (int i = 1; i <= numbers.Length; i++)  {  k = i;  combo = new int[k];  GenerateCombo(0);  }  List<int[]> finalList = new List<int[]>();  finalList = RemoveDuplicates(combinations);  foreach (int[] comb in finalList)  {  Print(comb);  }  }  private static void ProcessInput()  {  arraySize = int.Parse(Console.ReadLine());  numbers = new int[arraySize];  for (int i = 0; i < arraySize; i++)  {  numbers[i] = int.Parse(Console.ReadLine());  }  sum = int.Parse(Console.ReadLine());  }  static List<int[]> RemoveDuplicates(List<int[]> inputList)  {  List<int[]> finalList = new List<int[]>();  foreach (int[] current in inputList)  {  if (!Contains(finalList, current))  {  finalList.Add(current);  }  }  return finalList;  }  static bool Contains(List<int[]> list, int[] comparedValue)  {  foreach (int[] listValue in list)  {  if (listValue.SequenceEqual(comparedValue))  {  return true;  }  }  return false;  }  static void GenerateCombo(int startIndex)  {  if (startIndex >= k)  {  if (combo.Sum() == sum)  {  int[] tmpArr = new int[k];  Array.Copy(combo, tmpArr, k);  Array.Sort(tmpArr);  combinations.Add(tmpArr);  }  return;  }  for (int i = 0; i < arraySize; i++)  {  if (used[i] == 0)  {  used[i] = 1;  combo[startIndex] = numbers[i];  GenerateCombo(startIndex + 1);  used[i] = 0;  }  }  }  static void Print(int[] arr)  {  Console.Write("{");  for (int i = 0; i < arr.Length; i++)  {  Console.Write(" {0}", arr[i]);  }  Console.WriteLine(" }");  }  }  } | |
|  | |
| **Тестове** | |
| **Вход** | **Изход** |
| 4  2  3  1  -1  4 | { 1 3 }  { -1 2 3 } |
| **Вход** | **Изход** |
| 5  1  2  3  4  5  6 | { 1 5 }  { 2 4 }  { 1 2 3 } |

|  |  |
| --- | --- |
| Задача 9. Числа, чиято сума е N | |
| **Условие**  Даден е масив с цели **положителни** числа. Напишете програма, която проверява дали в масива съществуват едно или повече числа, чиято сума е **N**. | |
| **Описание на входа**  На първия ред е числото N – тоест сумата от числа в масива. На втория ред е масива като между всеки две числа има празен символ. | |
| **Описание на изхода**  Ако няма такова подмножество от числа чиято сума да е равна на N, то изхода е думата „No”. В противен случай се извеждат числата, чиято сума е N, като след всяко има празен символ. | |
| **Анализ на задачата** | |
| **Решение (сорс код)** | |
| using System;  using System.Collections.Generic;  using System.Linq;  namespace FindSubsetSumInArray  {  class Program  {  static void Main(string[] args)  {  int sumToFind = int.Parse(Console.ReadLine());  string inputArray = Console.ReadLine();  char[] splitters = new char[] { ' ', ',' };  string[] inputArraySplited = inputArray.Split(splitters, StringSplitOptions.RemoveEmptyEntries);  int[] numbersArray = new int[inputArraySplited.Length];  for (int i = 0; i < inputArraySplited.Length; i++)  {  numbersArray[i] = int.Parse(inputArraySplited[i]);  }  int subsetCombinations = (int)Math.Pow(2, numbersArray.Length);  int currentSum = 0;  bool isSubsetIsFound = false;  List<int> elementsInCurrentSubSet = new List<int>();  //position is zero-indexed  for (int currentCombination = 0; currentCombination < subsetCombinations; currentCombination++)  {  currentSum = 0;  for (int currentPosition = 0; currentPosition < numbersArray.Length; currentPosition++)  {  currentSum += (numbersArray[currentPosition] \* FindDigitOnPosition(currentCombination, currentPosition));  elementsInCurrentSubSet.Add(numbersArray[currentPosition] \* FindDigitOnPosition(currentCombination, currentPosition));  }  if (currentSum == sumToFind)  {  isSubsetIsFound = true;  //Print subset that has equal sum to searched sum  PrintSubset(sumToFind, elementsInCurrentSubSet);  break;  }  elementsInCurrentSubSet.Clear();  }  if (!isSubsetIsFound)  {  Console.WriteLine("No");  }  }  private static void PrintSubset(int sumToFind, List<int> elementsInCurrentSubSet)  {  elementsInCurrentSubSet.RemoveAll(item => item == 0);  for (int i = 0; i < elementsInCurrentSubSet.Count; i++)  {  Console.Write("{0} ", elementsInCurrentSubSet[i]);  }  Console.WriteLine();  }  static int FindDigitOnPosition(int number, int position)  {  int mask = 1;  mask <<= position;  mask &= number;  if (mask != 0)  {  return 1;  }  else  {  return 0;  }  }  }  } | |
| **Тестове** | |
| **Вход** | **Изход** |
| 14  -1 1 -2 -4 4 5 -2 6 | -1 4 5 6 |
| **Вход** | **Изход** |
| 6  1 5 2 3 4 5 -3 | 1 5 |

|  |  |
| --- | --- |
| Задача 10. Всички пътища между две клетки | |
| **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 първия ред на входа се прочитат две числа с един празен символ между тях – броя редове n и броя колони m на матрицата.  На следващите n реда се прочитат редовете на матрицата. Всеки ред се състои от m елемента. Всеки елемент е от тип char. За празна клетка в матрицата се използва символ точка - ‘.’, за непроходима – ‘\*’. За начална клетка – ‘s’ и за крайна клетка – ‘e’. | |
| **Описание на изхода**  Отпечатват се всички възможни пътища. За всеки възможен път се отпечатва дължината му и на следващите n реда се отпечатва матрицата, като клетките през които е минато са символа ‘p’. | |
| **Анализ на задачата**  Най-добрата възможност за реализирането на тази задач е Depth-First-Search(DFS). | |
| **Решение (сорс код)**  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace FindAllPaths  {  class Program  {  private static char[,] lab;  private static List<Tuple<int, int>> path = new List<Tuple<int, int>>();  static void Main(string[] args)  {  ReadInput();  int x;  int y;  FindStartLocation(out x, out y);  FindPathToExit(x, y);  }  private static void ReadInput()  {  string line = Console.ReadLine();  string[] rowsAndCols = line.Split(' ');  int rows = int.Parse(rowsAndCols[0]);  int cols = int.Parse(rowsAndCols[1]);  lab = new char[rows, cols];  for (int row = 0; row < rows; row++)  {  string rowStr = Console.ReadLine();  for (int col = 0; col < cols; col++)  {  lab[row, col] = rowStr[col];  }  }  }  private static void FindStartLocation(out int x, out int y)  {  x = -1;  y = -1;  for (int row = 0; row < lab.GetLength(0); row++)  {  for (int col = 0; col < lab.GetLength(1); col++)  {  if (lab[row, col] == 's')  {  x = row;  y = col;  lab[row, col] = '.';  return;  }  }  }  if (x == -1 || y == -1)  {  throw new ArgumentException("No starting location 's' found.");  }  }  static void FindPathToExit(int row, int col)  {  if (!InRange(row, col))  {  // We are out of the labyrinth -> can't find a path  return;  }  // Check if we have found the exit  if (lab[row, col] == 'e')  {  int pathLength = path.Count + 1;  Console.WriteLine(pathLength);  PrintPath(row, col);  }  if (lab[row, col] != '.')  {  // The current cell is not free -> can't find a path  return;  }  // Temporarily mark the current cell as visited  lab[row, col] = 'v';  path.Add(new Tuple<int, int>(row, col));  // Invoke recursion the explore all possible directions  FindPathToExit(row, col - 1); // left  FindPathToExit(row - 1, col); // up  FindPathToExit(row, col + 1); // right  FindPathToExit(row + 1, col); // down  // Mark back the current cell as free  lab[row, col] = '.';  path.RemoveAt(path.Count - 1);  }  static bool InRange(int row, int col)  {  bool rowInRange = row >= 0 && row < lab.GetLength(0);  bool colInRange = col >= 0 && col < lab.GetLength(1);  return rowInRange && colInRange;  }  static void PrintPath(int finalRow, int finalCol)  {  bool filledCell = false;  for (int row = 0; row < lab.GetLength(0); row++)  {  for (int col = 0; col < lab.GetLength(1); col++)  {  foreach (var cell in path)  {  if (row == cell.Item1 && col == cell.Item2)  {  Console.Write("r");  filledCell = true;  }  }  if (!filledCell)  {  if (lab[row, col] != '.')  {  Console.Write(lab[row, col]);  }  else  {  Console.Write(".");  }  }  filledCell = false;  }  Console.WriteLine();  }  }  }  } | |
|  | |
| **Тестове** | |
| **Вход** | **Изход** |
| 5 7  s..\*...  \*\*.\*.\*.  .......  .\*\*\*\*\*.  ......e | 15  rrr\*...  \*\*r\*.\*.  rrr....  r\*\*\*\*\*.  rrrrrre  15  rrr\*rrr  \*\*r\*r\*r  ..rrr.r  .\*\*\*\*\*r  ......e  11  rrr\*...  \*\*r\*.\*.  ..rrrrr  .\*\*\*\*\*r  ......e |
| **Вход** | **Изход** |
| 5 5  s....  ..\*\*\*  \*...\*  \*...e  \*\*\*\*\* | 8  rr...  .r\*\*\*  \*rrr\*  \*..re  \*\*\*\*\*  8  rr...  .r\*\*\*  \*rr.\*  \*.rre  \*\*\*\*\*  10  rr...  .r\*\*\*  \*rrr\*  \*rrre  \*\*\*\*\*  8  rr...  .r\*\*\*  \*r..\*  \*rrre  \*\*\*\*\*  8  r....  rr\*\*\*  \*rrr\*  \*..re  \*\*\*\*\*  8  r....  rr\*\*\*  \*rr.\*  \*.rre  \*\*\*\*\*  10  r....  rr\*\*\*  \*rrr\*  \*rrre  \*\*\*\*\*  8  r....  rr\*\*\*  \*r..\*  \*rrre  \*\*\*\*\* |

|  |  |
| --- | --- |
| Задача 11. Път между две клетки | |
| **Условие**  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. | |
| **Описание на входа**  На първия ред на входа се прочитат две числа с един празен символ между тях – броя редове n и броя колони m на матрицата.  На следващите n реда се прочитат редовете на матрицата. Всеки ред се състои от m елемента. Всеки елемент е от тип char. За празна клетка в матрицата се използва символ точка - ‘.’, за непроходима – ‘\*’. За начална клетка – ‘s’ и за крайна клетка – ‘e’. | |
| **Описание на изхода**  На единствения ред на изхода се изписва True или False в зависимост дали път съществува или не. | |
| **Анализ на задачата**  Подобна е на предишната – възможнти използвани алгоритми BFS или DFS | |
| **Решение (сорс код)**  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace CheckForPath  {  class Program  {  static char[,] lab;  static bool pathExists;  static void Main(string[] args)  {  ReadInput();  int x;  int y;  FindStartLocation(out x, out y);  pathExists = false;  FindPathToExit(x, y);  Console.WriteLine(pathExists);  }  private static void ReadInput()  {  string line = Console.ReadLine();  string[] rowsAndCols = line.Split(' ');  int rows = int.Parse(rowsAndCols[0]);  int cols = int.Parse(rowsAndCols[1]);  lab = new char[rows, cols];  for (int row = 0; row < rows; row++)  {  string rowStr = Console.ReadLine();  for (int col = 0; col < cols; col++)  {  lab[row, col] = rowStr[col];  }  }  }  private static void FindStartLocation(out int x, out int y)  {  x = -1;  y = -1;  for (int row = 0; row < lab.GetLength(0); row++)  {  for (int col = 0; col < lab.GetLength(1); col++)  {  if (lab[row, col] == 's')  {  x = row;  y = col;  lab[row, col] = '.';  return;  }  }  }  if (x == -1 || y == -1)  {  throw new ArgumentException("No starting location 's' found.");  }  }  static void FindPathToExit(int row, int col)  {  if (!InRange(row, col))  {  // We are out of the labyrinth -> can't find a path  return;  }  // Check if we have found the exit  if (lab[row, col] == 'e')  {  pathExists = true;  }  if (lab[row, col] != '.')  {  // The current cell is not free -> can't find a path  return;  }  // Temporarily mark the current cell as visited  lab[row, col] = 'v';  // Invoke recursion the explore all possible directions  if (!pathExists)  {  FindPathToExit(row, col - 1); // left  FindPathToExit(row - 1, col); // up  FindPathToExit(row, col + 1); // right  FindPathToExit(row + 1, col); // down  }  // Mark back the current cell as free  lab[row, col] = '.';  }  static bool InRange(int row, int col)  {  bool rowInRange = row >= 0 && row < lab.GetLength(0);  bool colInRange = col >= 0 && col < lab.GetLength(1);  return rowInRange && colInRange;  }  }  } | |
|  | |
| **Тестове** | |
| **Вход** | **Изход** |
| 5 7  s..\*...  \*\*.\*.\*.  .......  .\*\*\*\*\*.  ......e | True |
| **Вход** | **Изход** |
| 5 5  s....  .....  \*\*\*\*\*  .....  ....e | False |

|  |  |
| --- | --- |
| Задача 12. Най-дълга поредица от проходими елементи | |
| **Условие**  Напишете програма, която намира най-дългата поредица от съседни проходими клетки в матрица. | |
| **Описание на входа**  На първия ред на входа се прочитат две числа с един празен символ между тях – броя редове n и броя колони m на матрицата.  На следващите n реда се прочитат редовете на матрицата. Всеки ред се състои от m елемента. Всеки елемент е от тип char. За празна клетка в матрицата се използва символ точка - ‘.’, за непроходима – ‘\*’. | |
| **Описание на изхода**  На изхода се изписва: “The largest connected area is: X”, където X е максималния брой на свързани клетки. | |
| **Анализ на задачата**  Последователно се обхождат всички клетки, използвайки рекурсия и вариантите за възможна посока на пътя. | |
| **Решение (сорс код)**  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace LargestConnectedArea  {  class Program  {  private static char[,] lab;  private static int maxCount = 0;  private static int currentCount = 0;  static void Main(string[] args)  {  ReadInput();  for (int row = 0; row < lab.GetLength(0); row++)  {  for (int col = 0; col < lab.GetLength(1); col++)  {  if (lab[row, col] != '.')  {  continue;  }  else  {  currentCount = 0;  FindAllPaths(row, col);  if (currentCount > maxCount)  {  maxCount = currentCount;  }  }  }  }  Console.WriteLine("The largest connected area is: {0}", maxCount);  }  private static void ReadInput()  {  string line = Console.ReadLine();  string[] rowsAndCols = line.Split(' ');  int rows = int.Parse(rowsAndCols[0]);  int cols = int.Parse(rowsAndCols[1]);  lab = new char[rows, cols];  for (int row = 0; row < rows; row++)  {  string rowStr = Console.ReadLine();  for (int col = 0; col < cols; col++)  {  lab[row, col] = rowStr[col];  }  }  }  static void FindAllPaths(int row, int col)  {  if ((row < 0 || row >= lab.GetLength(0)) || (col < 0 || col >= lab.GetLength(1)))  {  return;  }  if (lab[row, col] == '.')  {  lab[row, col] = 'v';  currentCount++;  FindAllPaths(row, col + 1);  FindAllPaths(row + 1, col);  FindAllPaths(row, col - 1);  FindAllPaths(row - 1, col);  }  }  }  } | |
|  | |
| **Тестове** | |
| **Вход** | **Изход** |
| 6 7  ...\*...  \*\*.\*.\*.  ...\*...  \*\*\*\*\*\*.  ...\*\*..  ....... | The largest connected area is: 21 |
| **Вход** | **Изход** |
| 5 5  ...\*\*  \*\*.\*\*  ...\*\*  \*\*\*\*.  ..... | The largest connected area is: 7 |

|  |  |
| --- | --- |
| Задача 13. Площи от проходими елементи | |
| **Условие**  Даден е двумерен масив с проходими и непроходими клетки. Напишете програма, която намира всички площи съставени само от проходими клетки. | |
| **Описание на входа**  На първия ред на входа се прочитат две числа с един празен символ между тях – броя редове n и броя колони m на матрицата.  На следващите n реда се прочитат редовете на матрицата. Всеки ред се състои от m елемента. Всеки елемент е от тип char. За празна клетка в матрицата се използва символ точка - ‘.’, за непроходима – ‘\*’. | |
| **Описание на изхода**  За всяка площ се изписва на един ред колко клетки има в нея. На следващия ред се изреждат координатите на клетките оградени в скоби: (row, col). След всяка клетка има празен символ. | |
| **Анализ на задачата** | |
| **Решение (сорс код)**  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace Task10AllPassableAreas  {  class Program  {  private static char[,] lab;  private static List<Tuple<int, int>> path;  static void Main(string[] args)  {  ReadInput();  for (int row = 0; row < lab.GetLength(0); row++)  {  for (int col = 0; col < lab.GetLength(1); col++)  {  if (lab[row, col] != '.')  {  continue;  }  else  {  path = new List<Tuple<int, int>>();  FindAreas(row, col);  PrintArea(path);  }  }  }  }  private static void ReadInput()  {  string line = Console.ReadLine();  string[] rowsAndCols = line.Split(' ');  int rows = int.Parse(rowsAndCols[0]);  int cols = int.Parse(rowsAndCols[1]);  lab = new char[rows, cols];  for (int row = 0; row < rows; row++)  {  string rowStr = Console.ReadLine();  for (int col = 0; col < cols; col++)  {  lab[row, col] = rowStr[col];  }  }  }  static void FindAreas(int row, int col)  {  if ((row < 0 || row >= lab.GetLength(0)) || (col < 0 || col >= lab.GetLength(1)))  {  return;  }  if (lab[row, col] == '.')  {  lab[row, col] = 'v';  path.Add(new Tuple<int, int>(row, col));  FindAreas(row, col + 1);  FindAreas(row + 1, col);  FindAreas(row, col - 1);  FindAreas(row - 1, col);  }  }  static void PrintArea(List<Tuple<int, int>> area)  {  Console.WriteLine(area.Count);  foreach (var cell in area)  {  Console.Write("({0}, {1}) ", cell.Item1, cell.Item2);  }  Console.WriteLine();  }  }  } | |
|  | |
| **Тестове** | |
| **Вход** | **Изход** |
| 6 7  ...\*...  \*\*.\*.\*.  ...\*...  \*\*\*\*\*\*.  ...\*\*..  ....... | 7  (0, 0) (0, 1) (0, 2) (1, 2) (2, 2) (2, 1) (2, 0)  21  (0, 4) (0, 5) (0, 6) (1, 6) (2, 6) (3, 6) (4, 6) (5, 6) (5, 5) (5, 4) (5, 3) (5, 2) (5, 1) (5, 0) (4, 0) (4, 1) (4, 2) (4, 5) (2, 5) (2, 4) (1, 4) |
| **Вход** | **Изход** |
| 5 5  ...\*\*  \*\*.\*\*  ...\*\*  \*\*\*\*.  ..... | 7  (0, 0) (0, 1) (0, 2) (1, 2) (2, 2) (2, 1) (2, 0)  6  (3, 4) (4, 4) (4, 3) (4, 2) (4, 1) (4, 0) |

|  |  |
| --- | --- |
| Задача 14. Търсене на най-кратък път в лабиринт | |
| **Условие**  Реализирайте алгоритъма BFS (breath-first search) за търсене на най-кратък път в лаби­ринт. Ако се затруднявате, потърсете информация в Интернет. | |
| **Описание на входа**  На първия ред на входа се прочитат две числа с един празен символ между тях – броя редове n и броя колони m на матрицата.  На следващите n реда се прочитат редовете на матрицата. Всеки ред се състои от m елемента. Всеки елемент е от тип char. За празна клетка в матрицата се използва символа точка - ‘.’, за непроходима – ‘\*’. За начална клетка – ‘s’ и за крайна клетка – ‘e’. | |
| **Описание на изхода**  Ако има път се изписва единствено неговата дължина, ако няма се изписва “No”. | |
| **Анализ на задачата** | |
| **Решение (сорс код)**  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace Task9BFS  {  class Program  {  private static char[,] lab;  static Queue<Tuple<int, int>> q = new Queue<Tuple<int, int>>();  static Dictionary<Tuple<int, int>, Tuple<int, int>> parentTree = new Dictionary<Tuple<int, int>, Tuple<int, int>>();  static void Main(string[] args)  {  ReadInput();  int x;  int y;  char location = 's';  FindLocation(location, out x, out y);  q.Enqueue(new Tuple<int, int>(x, y));  BreadthFirstSearch();  }  private static void ReadInput()  {  string line = Console.ReadLine();  string[] rowsAndCols = line.Split(' ');  int rows = int.Parse(rowsAndCols[0]);  int cols = int.Parse(rowsAndCols[1]);  lab = new char[rows, cols];  for (int row = 0; row < rows; row++)  {  string rowStr = Console.ReadLine();  for (int col = 0; col < cols; col++)  {  lab[row, col] = rowStr[col];  }  }  }  private static void FindLocation(char location, out int x, out int y)  {  x = -1;  y = -1;  for (int row = 0; row < lab.GetLength(0); row++)  {  for (int col = 0; col < lab.GetLength(1); col++)  {  if (lab[row, col] == location)  {  x = row;  y = col;  lab[row, col] = ' ';  return;  }  }  }  if (x == -1 || y == -1)  {  throw new ArgumentException("No starting location 's' found.");  }  }  static void BreadthFirstSearch()  {  Tuple<int, int> cell = q.Dequeue();  if (lab[cell.Item1, cell.Item2] == 'e')  {  CalculatePathLength();  return;  }  int qCount = q.Count;  lab[cell.Item1, cell.Item2] = 's';  if (InRange(cell.Item1, cell.Item2 + 1) && lab[cell.Item1, cell.Item2 + 1] != '\*' && lab[cell.Item1, cell.Item2 + 1] != 's')  {  q.Enqueue(new Tuple<int, int>(cell.Item1, cell.Item2 + 1));  }  if (InRange(cell.Item1 + 1, cell.Item2) && lab[cell.Item1 + 1, cell.Item2] != '\*' && lab[cell.Item1 + 1, cell.Item2] != 's')  {  q.Enqueue(new Tuple<int, int>(cell.Item1 + 1, cell.Item2));  }  if (InRange(cell.Item1 - 1, cell.Item2) && lab[cell.Item1 - 1, cell.Item2] != '\*' && lab[cell.Item1 - 1, cell.Item2] != 's')  {  q.Enqueue(new Tuple<int, int>(cell.Item1 - 1, cell.Item2));  }  if (InRange(cell.Item1, cell.Item2 - 1) && lab[cell.Item1, cell.Item2 - 1] != '\*' && lab[cell.Item1, cell.Item2 - 1] != 's')  {  q.Enqueue(new Tuple<int, int>(cell.Item1, cell.Item2 - 1));  }  if (q.Count == 0)  {  Console.WriteLine("No");  return;  }  foreach (var newCell in q)  {  if (!parentTree.ContainsKey(newCell))  {  parentTree.Add(newCell, cell);  }  }  BreadthFirstSearch();  }  static void CalculatePathLength()  {  int x;  int y;  char location = 'e';  FindLocation(location, out x, out y);  Tuple<int, int> cell = new Tuple<int, int>(x, y);  int pathLength = 0;  Tuple<int, int> parentCell = parentTree[cell];  while (parentTree.ContainsKey(parentCell))  {  cell = parentCell;  parentCell = parentTree[cell];  pathLength++;  }  pathLength++;  Console.WriteLine(pathLength);  }  static bool InRange(int row, int col)  {  bool rowInRange = row >= 0 && row < lab.GetLength(0);  bool colInRange = col >= 0 && col < lab.GetLength(1);  return rowInRange && colInRange;  }  }  } | |
|  | |
| **Тестове** | |
| **Вход** | **Изход** |
| 5 7  s..\*...  \*\*.\*.\*.  .......  .\*\*\*\*\*.  ......e | 10 |
| **Вход** | **Изход** |
| 5 5  s....  .....  \*\*\*\*\*  .....  ....e | No |

|  |  |
| --- | --- |
| Задача 15. Обхождане на хард-диска | |
| **Условие**  Напишете рекурсивна програма, която обхожда целия твърд диск **C:\** рекурсивно и отпечатва всички папки и файловете в тях. | |
| **Описание на входа**  На входа се въвежда пълната пътека към директорията, която ще обхождаме. | |
| **Описание на изхода**  За всяка поддиректория се изписва “Found dir: пълното име на директорията“, за всеки файл: “Found file: пълно име на файла“. | |
| **Анализ на задачата** | |
| **Решение (сорс код)**  using System;  using System.IO;  namespace Task11DisplayFiles  {  class Program  {  static void Main(string[] args)  {  string path = Console.ReadLine();  TraverseDirs(new DirectoryInfo(path));  }  static void TraverseDirs(DirectoryInfo dir)  {  try  {  foreach (DirectoryInfo iInfo in dir.GetDirectories())  {  Console.WriteLine("Found dir: " + iInfo.FullName);  TraverseDirs(iInfo);  }  }  catch (Exception)  {  }  try  {  foreach (FileInfo iInfo in dir.GetFiles())  {  Console.WriteLine("Found file: " + iInfo.FullName);  }  }  catch (Exception)  {  }  }  }  } | |
|  | |
| **Тестове** | |
| **Вход** | **Изход** |
| C:\Windows\Media\Afternoon | Found file: C:\Windows\Media\Afternoon\Desktop.ini  Found file: C:\Windows\Media\Afternoon\Windows Balloon.wav  Found file: C:\Windows\Media\Afternoon\Windows Battery Critical.wav  Found file: C:\Windows\Media\Afternoon\Windows Battery Low.wav  Found file: C:\Windows\Media\Afternoon\Windows Critical Stop.wav  Found file: C:\Windows\Media\Afternoon\Windows Default.wav  Found file: C:\Windows\Media\Afternoon\Windows Ding.wav  Found file: C:\Windows\Media\Afternoon\Windows Error.wav  Found file: C:\Windows\Media\Afternoon\Windows Exclamation.wav  Found file: C:\Windows\Media\Afternoon\Windows Feed Discovered.wav  Found file: C:\Windows\Media\Afternoon\Windows Hardware Fail.wav  Found file: C:\Windows\Media\Afternoon\Windows Hardware Insert.wav  Found file: C:\Windows\Media\Afternoon\Windows Hardware Remove.wav  Found file: C:\Windows\Media\Afternoon\Windows Information Bar.wav  Found file: C:\Windows\Media\Afternoon\Windows Logoff Sound.wav  Found file: C:\Windows\Media\Afternoon\Windows Logon Sound.wav  Found file: C:\Windows\Media\Afternoon\Windows Navigation Start.wav  Found file: C:\Windows\Media\Afternoon\Windows Notify.wav  Found file: C:\Windows\Media\Afternoon\Windows Pop-up Blocked.wav  Found file: C:\Windows\Media\Afternoon\Windows Print complete.wav  Found file: C:\Windows\Media\Afternoon\Windows User Account Control.wav |
| **Вход** | **Изход** |
| C:\Windows\Media\Sonata | Found file: C:\Windows\Media\Sonata\Desktop.ini  Found file: C:\Windows\Media\Sonata\Windows Balloon.wav  Found file: C:\Windows\Media\Sonata\Windows Battery Critical.wav  Found file: C:\Windows\Media\Sonata\Windows Battery Low.wav  Found file: C:\Windows\Media\Sonata\Windows Critical Stop.wav  Found file: C:\Windows\Media\Sonata\Windows Default.wav  Found file: C:\Windows\Media\Sonata\Windows Ding.wav  Found file: C:\Windows\Media\Sonata\Windows Error.wav  Found file: C:\Windows\Media\Sonata\Windows Exclamation.wav  Found file: C:\Windows\Media\Sonata\Windows Feed Discovered.wav  Found file: C:\Windows\Media\Sonata\Windows Hardware Fail.wav  Found file: C:\Windows\Media\Sonata\Windows Hardware Insert.wav  Found file: C:\Windows\Media\Sonata\Windows Hardware Remove.wav  Found file: C:\Windows\Media\Sonata\Windows Information Bar.wav  Found file: C:\Windows\Media\Sonata\Windows Logoff Sound.wav  Found file: C:\Windows\Media\Sonata\Windows Logon Sound.wav  Found file: C:\Windows\Media\Sonata\Windows Navigation Start.wav  Found file: C:\Windows\Media\Sonata\Windows Notify.wav  Found file: C:\Windows\Media\Sonata\Windows Pop-up Blocked.wav  Found file: C:\Windows\Media\Sonata\Windows Print complete.wav  Found file: C:\Windows\Media\Sonata\Windows User Account Control.wav |