# Exercises: BFS and DFS

Problems for exercises and homework for the ["Algorithms" course @ SoftUni](https://softuni.bg/opencourses/algorithms).

You can check your solutions here: <https://judge.softuni.bg/Contests/1560/Basic-Algorithms-Exercise>

## Reverse Array

Write a program that reverses and prints an array. Use **recursion**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 2 3 4 5 6 | 6 5 4 3 2 1 |

## Nested Loops To Recursion

Write a program that simulates the execution of n nested loops **from 1 to n** which prints the values of all its iteration variables at any given time on a single line. **Use recursion.**

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Solution with nested loops**  **(assuming n is positive)** |
| 2 | 1 1  1 2  2 1  2 2 | int n = 2;  for (int i1 = 1; i1 <= n; i1++)  {  for (int i2 = 1; i2 <= n; i2++)  {  Console.WriteLine($"{i1} {i2}");  }  } |
| 3 | 1 1 1 1 1 2  1 1 3  1 2 1  1 2 2  …  3 2 3  3 3 1  3 3 2  3 3 3 | int n = 3;  for (int i1 = 1; i1 <= n; i1++)  {  for (int i2 = 1; i2 <= n; i2++)  {  for (int i3 = 1; i3 <= n; i3++)  {  Console.WriteLine($"{i1} {i2} {i3}");  }  }  } |

## Connected Areas in a Matrix

Let’s define a **connected area** in a matrix as an area of cells in which there is a **path between every two cells**.

Write a program to find **all** connected areas in a matrix.

On the console, print the **total number of areas found**, and on a separate line some info about each of the areas – its position (top-left corner) and size.

**Order** the areas by size (in descending order) so that the **largest area is printed first**. If several areas have the same size, order them **by their position**, first by the row, then by the column of the top-left corner. So, if there are two connected areas with the same size, the one which is above and/or to the left of the other will be printed first.

On the first line, you will get the **number of rows**.

On the second line, you will get the **number of columns**.

The rest of the input will be the **actual matrix**.

### Examples

|  |  |
| --- | --- |
| **Example Layout** | **Output** |
| 4  9   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | - | - | - | \* | - | - | - | \* | - | | - | - | - | \* | - | - | - | \* | - | | - | - | - | \* | - | - | - | \* | - | | - | - | - | - | \* | - | \* | - | - | | Total areas found: 3  Area #1 at (0, 0), size: 13  Area #2 at (0, 4), size: 10  Area #3 at (0, 8), size: 5 |
| 5  10   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | \* | - | - | \* | - | - | - | \* | - | - | | \* | - | - | \* | - | - | - | \* | - | - | | \* | - | - | \* | \* | \* | \* | \* | - | - | | \* | - | - | \* | - | - | - | \* | - | - | | \* | - | - | \* | - | - | - | \* | - | - | | Total areas found: 4  Area #1 at (0, 1), size: 10  Area #2 at (0, 8), size: 10  Area #3 at (0, 4), size: 6  Area #4 at (3, 4), size: 6 |

### Hints

* Create a method to find the first traversable cell which hasn’t been visited. This would be the top-left corner of a connected area. If there is no such cell, this means all areas have been found.
* You can create a class to hold info about a connected area (its position and size). Additionally, you can implement IComparable and store all areas found in a SortedSet.

## The Matrix

You are given a matrix (2D array) of lowercase alphanumeric characters (a-z, 0-9), a starting position – defined by a start row startRow and a start column startCol – and a filling symbol fillChar. Let’s call the symbol originally at startRow and startCol the startChar. Write a program, which, starting from the symbol at startRow and startCol, changes to fillChar every symbol in the matrix which:

* is equal to startChar AND
* can be reached from startChar by going up (row – 1), down (row + 1), left (col – 1) and right (col + 1) and “stepping” ONLY on symbols equal startChar

So, you basically start from startRow and startCol and can move either by changing the row OR column (not both at once, i.e. you can’t go diagonally) by 1, and can only go to positions which have the startChar written on them. Once you find all those positions, you change them to fillChar.

In other words, you need to implement something like the Fill tool in MS Paint, but for a 2D char array instead of a bitmap.

### Input

On the first line, two integers will be entered – the number R of rows and number C of columns.

On each of the next R lines, C characters separated by single spaces will be entered – the symbols of the Rth row of the matrix, starting from the 0th column and ending at the C-1 column.

On the next line, a single character – the fillChar – will be entered.

On the last line, two integers – startRow and startCol – separated by a single space, will be entered.

### Output

The output should consist of R lines, each consisting of exactly C characters, **NOT SEPARATED** by spaces, representing the matrix after the fill operation has been finished.

### Constraints

0 < R, C < 20   
0 <= startRow < R   
0 <= startCol < C

All symbols in the input matrix will be lowercase alphanumerics (a-z, 0-9). The fillChar will also be alphanumeric and lowercase.

The total running time of your program should be no more than 0.1s

The total memory allowed for use by your program is 5MB

### Examples

|  |  |
| --- | --- |
| **Example Input** | **Expected Output** |
| 5 3  a a a  a a a  a b a  a b a  a b a  x  0 0 | xxx  xxx  xbx  xbx  xbx |
| 5 3  a a a  a a a  a b a  a b a  a b a  x  2 1 | aaa  aaa  axa  axa  axa |
| 5 6  o o 1 1 o o  o 1 o o 1 o  1 o o o o 1  o 1 o o 1 o  o o 1 1 o o  3  2 1 | oo11oo  o1331o  133331  o1331o  oo11oo |
| 5 6  o o o o o o  o o o 1 o o  o o 1 o 1 1  o 1 1 w 1 o  1 o o o o o  z  4 1 | oooooo  ooo1oo  oo1o11  o11w1z  1zzzzz |
| 5 6  o 1 o o 1 o  o 1 o o 1 o  o 1 1 1 1 o  o 1 o w 1 o  o o o o o o  z  4 0 | z1oo1z  z1oo1z  z1111z  z1zw1z  zzzzzz |

**Hints**

For some of the tests you can solve the problem with naive approach, however complete solution can be obtained by using **Stack**, **Queue**, **DFS** or **BFS** – go search on the internet.

## Longest Increasing Subsequence (LIS)

Read a **list of integers** and find the **longest increasing subsequence** (LIS). If several such exist, print the **leftmost**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| **1** | 1 |
| 7 **3 5** 8 -1 0 **6 7** | 3 5 6 7 |
| **1 2** 5 **3 5** 2 4 1 | 1 2 3 5 |
| **0** 10 20 30 30 40 **1** 50 **2 3 4 5 6** | 0 1 2 3 4 5 6 |
| 11 12 13 **3** 14 **4** 15 **5 6 7 8** 7 **16** 9 8 | 3 4 5 6 7 8 16 |
| **3** 14 **5** 12 15 **7 8 9 11** 10 1 | 3 5 7 8 9 11 |

### Hints

* Assume we have n numbers in an array nums[0…n-1].
* Let len[p] holds the length of the longest increasing subsequence (LIS) ending at position p.
* In a for loop, we shall calculate len[p] for p = 0 … n-1 as follows:
  + Let left is the leftmost position on the left of p (left < p), such that len[left] is the largest possible.
  + Then, len[p] = 1 + len[left]. If left does not exist, len[p] = 1.
  + Also, save prev[p] = left (we hold if prev[] the previous position, used to obtain the best length for position p).
* Once the values for len[0…n-1] are calculated, restore the LIS starting from position p such that len[p] is maximal and go back and back through p = prev[p].
* The table below illustrates these computations:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| nums[] | **3** | **14** | **5** | **12** | **15** | **7** | **8** | **9** | **11** | **10** | **1** |
| len[] | 1 | 2 | 2 | 3 | 4 | 3 | 4 | 5 | 6 | 6 | 1 |
| prev[] | -1 | 0 | 0 | 2 | 3 | 2 | 5 | 6 | 7 | 7 | -1 |
| LIS | {3} | {3,14} | {3,5} | {3,5,12} | {3,5,12,15} | {3,5,7} | {3,5,7,8} | {3,5,7,8,9} | {3,5,7,8,9,11} | {3,5,7,8,9,10} | {1} |