**Exercise: Graph Theory Traversal and Shortest Paths**

This document defines the lab for the ["Algorithms – Fundamentals (Java)" course @ Software University](https://softuni.bg/trainings/3811/algorithms-fundamentals-with-java-june-2022). Please submit your solutions (source code) to all below-described problems in [Judge](https://judge.softuni.bg/Contests/2464/Graph-Theory-Traversal-and-Shortest-Paths-Exercise).

## Distance between Vertices

We are given a **directed graph**. We are given also a set of **pairs of vertices**. Find the **shortest distance between each pair** of vertices or **-1** if there is no path connecting them.

On the first line, you will get **N**, the number of vertices in the graph. On the second line, you will get P, the number of pairs between which to find the shortest distance.

On the next **N,** lines will be the edges of the graph and on the next **P** lines, the pairs.

Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Picture** | **Output** |
| 2  2  1:2  2:  1-2  2-1 |  | {1, 2} -> 1  {2, 1} -> -1 |
| 8  4  1:4  2:4  3:4 5  4:6  5:3 7 8  6:  7:8  8:  1-6  1-5  5-6  5-8 |  | {1, 6} -> 2  {1, 5} -> -1  {5, 6} -> 3  {5, 8} -> 1 |
| 9  8  11:4  4:12 1  1:12 21 7  7:21  12:4 19  19:1 21  21:14 31  14:14  31:  11-7  11-21  21-4  19-14  1-4  1-11  31-21  11-14 |  | {11, 7} -> 3  {11, 21} -> 3  {21, 4} -> -1  {19, 14} -> 2  {1, 4} -> 2  {1, 11} -> -1  {31, 21} -> -1  {11, 14} -> 4 |

Hint

For each pair use **BFS** to find all paths from the source to the destination vertex.

## Areas in Matrix

We are given a matrix of letters of size N \* M. Two cells are neighbors if they share a common wall. Write a program to find the connected areas of neighbor cells holding the same letter. Display the **total number of areas** and the number of **areas for each alphabetical letter** (ordered by alphabetical order).

On the **first line** is given the **number of rows**.

Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Picture** | **Output** |
| 6  aacccaac  baaaaccc  baabaccc  bbdaaccc  ccdccccc  ccdccccc |  | Areas: 8  Letter 'a' -> 2  Letter 'b' -> 2  Letter 'c' -> 3  Letter 'd' -> 1 |
| 3  aaa  aaa  aaa |  | Areas: 1  Letter 'a' -> 1 |
| 5  asssaadas  adsdasdad  sdsdadsas  sdasdsdsa  ssssasddd |  | Areas: 21  Letter 'a' -> 6  Letter 'd' -> 7  Letter 's' -> 8 |

### Hint

Initially mark all cells as **unvisited**. Start a **recursive DFS traversal** (or BFS) from each unvisited cell and mark all reached cells as visited. Each DFS traversal will find one of the **connected areas**.

## Cycles in a Graph

Write a program to check whether an undirected graph is **acyclic** or holds any cycles. The input ends with the "**End**" line.

Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Picture** | **Output** |
| C-G  End |  | Acyclic: Yes |
| A-F  F-D  D-A  End |  | Acyclic: No |
| E-Q  Q-P  P-B  End |  | Acyclic: Yes |
| K-J  J-N  N-L  N-M  M-I  End |  | Acyclic: Yes |
| K-X  X-Y  X-N  N-J  M-N  A-Z  B-P  I-F  A-Y  Y-L  M-I  F-P  Z-E  P-E  End |  | Acyclic: No |

Hint

Modify the Topological Sorting algorithm (source removal or DFS-based).

## Salaries

We have a **hierarchy** between the employees in a company. Employees can have one or several direct managers. People who **manage nobody** are called **regular employees** and their salaries are **1**. People who manage at least one employee are called **managers**. Each manager takes a **salary** that is equal to the **sum of the salaries of their directly managed employees**. Managers cannot manage directly or indirectly (transitively) themselves. Some employees might have no manager (like the big boss). See a sample hierarchy in a company along with the salaries computed following the above-described rule:



In the above example, employees 0 and 3 are regular employees and take salary 1. All others are managers and take the sum of the salaries of their directly managed employees. For example, manager 1 takes salary 3 + 2 + 1 = 6 (sum of the salaries of employees 2, 5 and 0). In the above example employees, 4 and 1 have no manager.

If we have **N** employees, they will be indexed from 0 to N – 1. For each employee, you’ll be given a string with N symbols. The symbol at given index **i**, either **'Y' or 'N'**, shows whether the current employee is a direct manager of employee **i**.

**Hint**: find the node with no parent and start a **DFS traversal** from it to calculate the salaries on the tree recursively.

### Input

* The input data should be read from the console.
* On the first line, you’ll be given an integer N.
* On the next N lines, you’ll be given strings with N symbols (either 'Y' or 'N').
* The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

* The output should be printed on the console. It should consist of one line.
* On the only output line print the sum of the salaries of all employees.

### Constraints

* N will be an integer in the range [1 … 50].
* For each i-th line, the i-th symbol will be 'N'.
* If employee A is the manager of employee B, B will not be a manager of A.
* Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 1  N | 1 | Only 1 employee with salary 1. |
| 4  NNYN  NNYN  NNNN  NYYN | 5 | We have 4 employees. 0, 1, and 3 are managers of 2. 3 is also a manager of 1. Therefore:  salary(2) = 1  salary(0) = salary(2) = 1  salary(1) = salary(2) = 1  salary(3) = salary(2) + salary(1) = 2 |
| 6  NNNNNN  YNYNNY  YNNNNY  NNNNNN  YNYNNN  YNNYNN | 17 |  |

## Break Cycles

You are given an **undirected multi-graph**. Remove a minimal number of edges to **make the graph acyclic** (to break all cycles). As a result, print the number of edges removed and the removed edges. If several edges can be removed to break a certain cycle, remove the smallest of them in alphabetical order (smallest start vertex in alphabetical order and smallest end vertex in alphabetical order).

Examples

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Picture** | **Output** | **Picture After Removal** |
| 1 -> 2 5 4  2 -> 1 3  3 -> 2 5  4 -> 1  5 -> 1 3  6 -> 7 8  7 -> 6 8  8 -> 6 7 |  | Edges to remove: 2  1 - 2  6 - 7 |  |
| K -> X J  J -> K N  N -> J X L M  X -> K N Y  M -> N I  Y -> X L  L -> N I Y  I -> M L  A -> Z Z Z  Z -> A A A  F -> E B P  E -> F P  P -> B F E  B -> F P |  | Edgeds to remove: 7  A - Z  A - Z  B - F  E - F  I - L  J - K  L - N |  |

### Hint

* Enumerate edges {**s**, **e**} in alphabetical order. For each edge {**s**, **e**} check whether it closes a cycle. If yes - remove it.
  + To check whether an edge {**s**, **e**} closes a cycle, temporarily remove the edge {s, e} and then try to find a path from **s** to **e** using DFS or BFS.

## 6. Road Reconstruction

You have to rebuild some roads in your city. Write a program that finds all the roads without which **buildings** in the city will become **unreachable**. You will receive how many **buildings** the town has on the first line, then you will receive the number of **streets** and finally, for **each street,** you will receive which **buildings it connects**. Find all the streets that are important and **cannot be removed** and print them as shown in the examples.

### Input

* On the first line, you will receive the **amount** of the **buildings.**
* On the second line, you will receive the **amount** of the **streets** (**n**).
* On the next **"n"** lines you will receive which **buildings** each **street connects.**

### Output

* On the first line print: **"Important streets:"**
* On the next lines (if any) print the street in the format: **"{firstBuilding} {secondBuilding}"**
* The **order** of the output does **not matter** as long as you print all of the important streets.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  5  1 - 0  0 - 2  2 - 1  0 - 3  3 - 4 | Important streets:  3 4  0 3 |
| 7  8  0 - 1  1 - 2  2 - 0  1 - 3  1 - 4  1 - 6  3 - 5  4 - 5 | Important streets:  1 6 |

## 12. 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 that 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

|  |  |
| --- | --- |
| Input | 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 a naive approach, however, a complete solution can be obtained by using **Stack**, **Queue**, **DFS,** or **BFS.**