# Exercise: Graph Theory, Traversal and Topological Sorting

This document defines the exercise for ["Algorithms with Python" course @Software University](https://softuni.bg/opencourses/algorithms-with-python)".

Please submit your solutions (source code) to all below-described problems in [Judge](https://judge.softuni.org/Contests/3463/).

## 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  8  aacccaac  baaaaccc  baabaccc  bbdaaccc  ccdccccc  ccdccccc |  | Areas: 8  Letter 'a' -> 2  Letter 'b' -> 2  Letter 'c' -> 3  Letter 'd' -> 1 |
| 3  3  aaa  aaa  aaa |  | Areas: 1  Letter 'a' -> 1 |
| 5  9  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 a directed graph is **acyclic** or holds any cycles. The input ends with "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 |

## 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** which is equal to the **sum of the salaries of their directly managed employees**.

Managers cannot manage directly or indirectly themselves. Some employees might have no manager. 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 will be given a string with N symbols. The symbols are either **'Y' or 'N'**, showing all employees that are managed by the current employee.

### Input

* On the first line, you will be given an integer N.
* On the next N lines, you will 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

* Print the sum of the salaries of all employees.

### Constraints

* N will be an integer in the range [1 … 50].
* If employee A is the manager of employee B, B will not be a manager of A.

### 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** |
| 8  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 |  |
| 14  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 |  | Edges to remove: 7  A - Z  A - Z  B - F  E - F  I - L  J - K  L - N |  |

### Hint

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

## Road Reconstruction

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 in ascending order (e. g. 3 0 should be printed as 0 3).

### Input

* On the first line, you will receive the **number** of **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** if you print all the important streets.

### Examples

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