Python Training Code Assignments:

Problem 1

Two Sums

Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to* `target`.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

Example 1:

```
Input: nums = [2,7,11,15], target = 9

Output: [0,1]

Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
```

Example 2:

```
Input: nums = [3,2,4], target = 6

Output: [1,2]
```

Example 3:

```
Input: nums = [3,3], target = 6

Output: [0,1]
```

Constraints:

- `2 <= nums.length <= 10⁴`
- `-10⁹ <= nums[i] <= 10⁹`
- `-10⁹ <= target <= 10⁹`
- **Only one valid answer exists.**

Code to start with:

```python
def twoSum(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: List[int]
        """
```

Problem 2

Combination Sum

Given an array of **distinct** integers `candidates` and a target integer `target`, return *a list of all **unique combinations** of* `candidates` *where the chosen numbers sum to* `target`. You may return the combinations in **any order**.

The **same** number may be chosen from `candidates` an **unlimited number of times**. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

It is **guaranteed** that the number of unique combinations that sum up to `target` is less than `150` combinations for the given input.

Example 1:

```
Input: candidates = [2,3,6,7], target = 7

Output: [[2,2,3],[7]]

Explanation:

2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.

7 is a candidate, and 7 = 7.

These are the only two combinations.
```

Example 2:

```
Input: candidates = [2,3,5], target = 8

Output: [[2,2,2,2],[2,3,3],[3,5]]
```

Example 3:

```
Input: candidates = [2], target = 1

Output: []
```

Constraints:

- `1 <= candidates.length <= 30`
- `1 <= candidates[i] <= 200`
- All elements of `candidates` are **distinct**.
- `1 <= target <= 500`

```python
def combinationSum(self, candidates, target):
    """
    :type candidates: List[int]
    :type target: int
    :rtype: List[List[int]]
    """
```

Problem 3

Roman number to Integers

Roman numerals are represented by seven different symbols: `I`, `V`, `X`, `L`, `C`, `D` and `M`.

```
Symbol       Value

I             1

V             5

X             10

L             50

C             100

D             500

M             1000
```

For example, `2` is written as `II` in Roman numeral, just two one's added together. `12` is written as `XII`, which is simply `X + II`. The number `27` is written as `XXVII`, which is `XX + V + II`.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not `IIII`. Instead, the number four is written as `IV`. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as `IX`. There are six instances where subtraction is used:

- `I` can be placed before `V` (5) and `X` (10) to make 4 and 9.
- `X` can be placed before `L` (50) and `C` (100) to make 40 and 90.
- `C` can be placed before `D` (500) and `M` (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer.

Example 1:

```
Input: s = "III"

Output: 3

Explanation: III = 3.
```

Example 2:

```
Input: s = "LVIII"

Output: 58

Explanation: L = 50, V= 5, III = 3.
```

Example 3:

```
Input: s = "MCMXCIV"
```

Output: 1994

Explanation: M = 1000, CM = 900, XC = 90 and IV = 4.

Constraints:

- `1 <= s.length <= 15`
- `s` contains only the characters `('I', 'V', 'X', 'L', 'C', 'D', 'M')`.
- It is **guaranteed** that `s` is a valid roman numeral in the range `[1, 3999]`.

```
def romanToInt(self, s):
    """
    :type s: str
    :rtype: int
    """
```

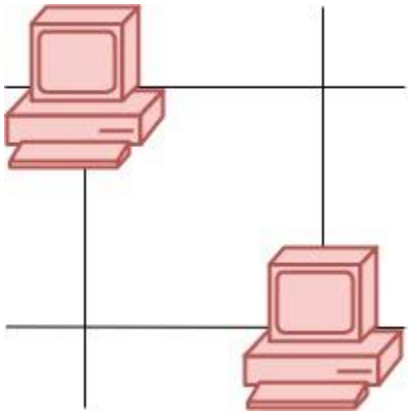
Problem 4

Count Servers that Communicate

You are given a map of a server centre, represented as a `m * n` integer matrix `grid`, where 1 means that on that cell there is a server and 0 means that it is no server. Two servers are said to communicate if they are on the same row or on the same column.

Return the number of servers that communicate with any other server.

Example 1:

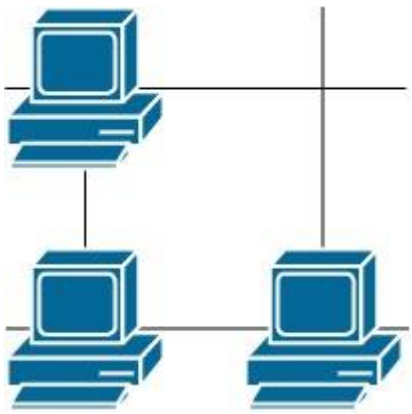


```
Input: grid = [[1,0], [0,1]]

Output: 0

Explanation: No servers can communicate with others.
```

Example 2:

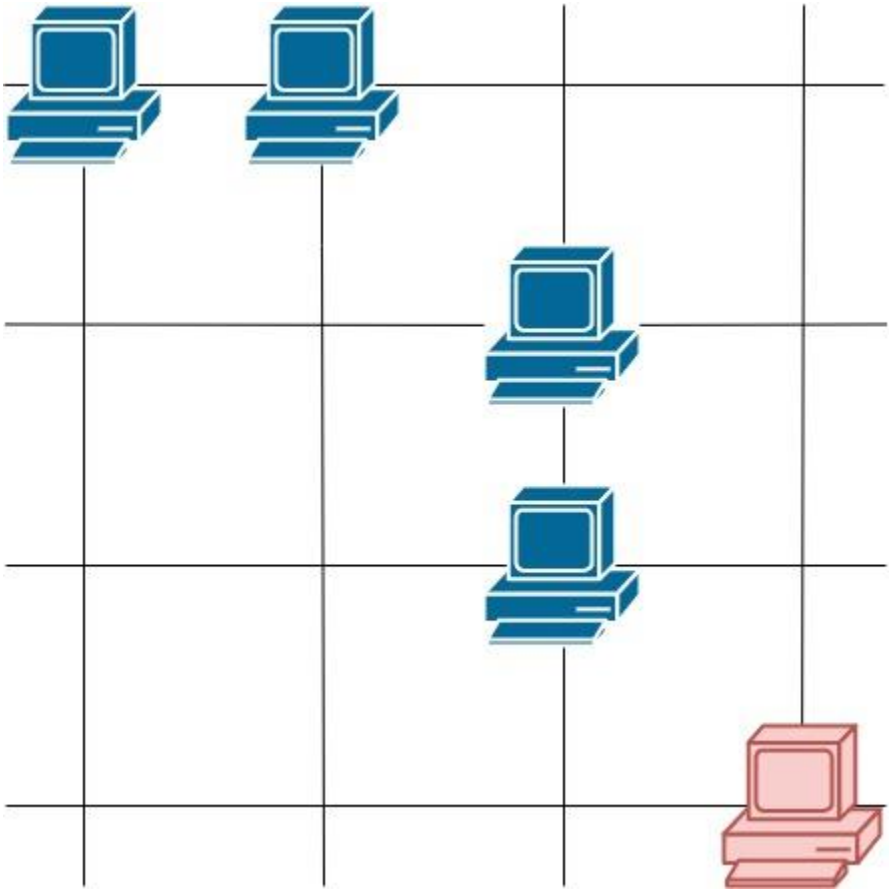


```
Input: grid = [[1,0],[1,1]]

Output: 3

Explanation: All three servers can communicate with at least one other server.
```

Example 3:



Input: grid = [[1,1,0,0],[0,0,1,0],[0,0,1,0],[0,0,0,1]]

Output: 4

Explanation: The two servers in the first row can communicate with each other. The two servers in the third column can communicate with each other. The server at right bottom corner can't communicate with any other server.

Constraints:

- `m == grid.length`
- `n == grid[i].length`
- `1 <= m <= 250`
- `1 <= n <= 250`
- `grid[i][j] == 0 or 1`

```
class Solution(object):
    def countServers(self, grid):
        """
        :type grid: List[List[int]]
        :rtype: int
        """
```