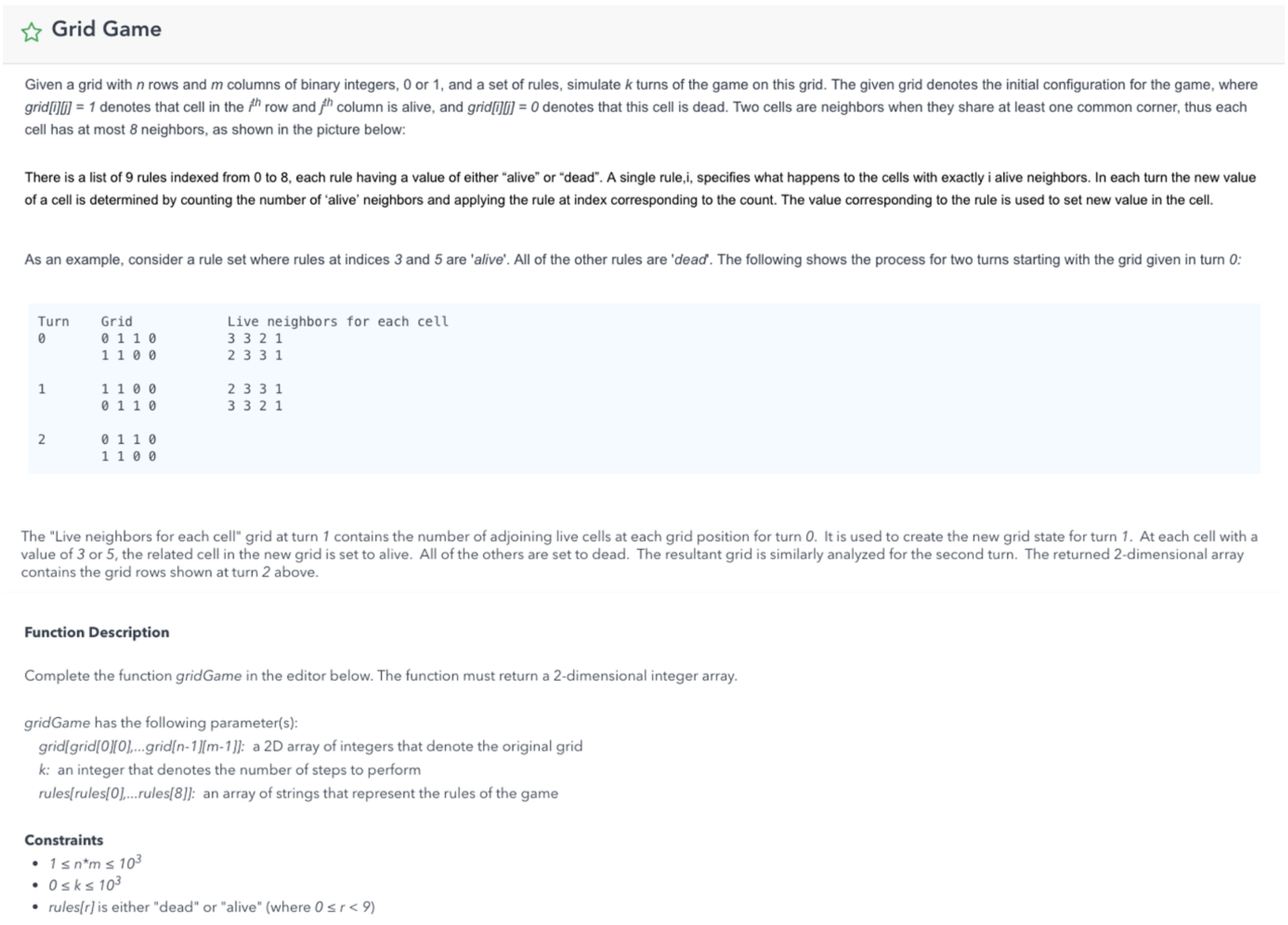
Question 1.



Grid Game

Given a grid with *n* rows and *m* columns of binary integers, 0 or 1, and a set of rules, simulate *k* turns of the game on this grid. The given grid denotes the initial configuration for the game, where *grid[i][j] = 1* denotes that cell in the i^{th} row and j^{th} column is alive, and *grid[i][j] = 0* denotes that this cell is dead. Two cells are neighbors when they share at least one common corner, thus each cell has at most *8* neighbors, as shown in the picture below:

There is a list of 9 rules indexed from 0 to 8, each rule having a value of either "alive" or "dead". A single rule,i, specifies what happens to the cells with exactly i alive neighbors. In each turn the new value of a cell is determined by counting the number of 'alive' neighbors and applying the rule at index corresponding to the count. The value corresponding to the rule is used to set new value in the cell.

As an example, consider a rule set where rules at indices *3* and *5* are '*alive*'. All of the other rules are '*dead*'. The following shows the process for two turns starting with the grid given in turn *0:*

```
Turn    Grid            Live neighbors for each cell
0       0 1 1 0         3 3 2 1
        1 1 0 0         2 3 3 1

1       1 1 0 0         2 3 3 1
        0 1 1 0         3 3 2 1

2       0 1 1 0
        1 1 0 0
```

The "Live neighbors for each cell" grid at turn *1* contains the number of adjoining live cells at each grid position for turn *0*. It is used to create the new grid state for turn *1*. At each cell with a value of *3* or *5*, the related cell in the new grid is set to alive. All of the others are set to dead. The resultant grid is similarly analyzed for the second turn. The returned 2-dimensional array contains the grid rows shown at turn *2* above.

Function Description

Complete the function *gridGame* in the editor below. The function must return a 2-dimensional integer array.

gridGame has the following parameter(s):
grid[grid[0][0],...grid[n-1][m-1]]: a 2D array of integers that denote the original grid
k: an integer that denotes the number of steps to perform
rules[rules[0],...rules[8]]: an array of strings that represent the rules of the game

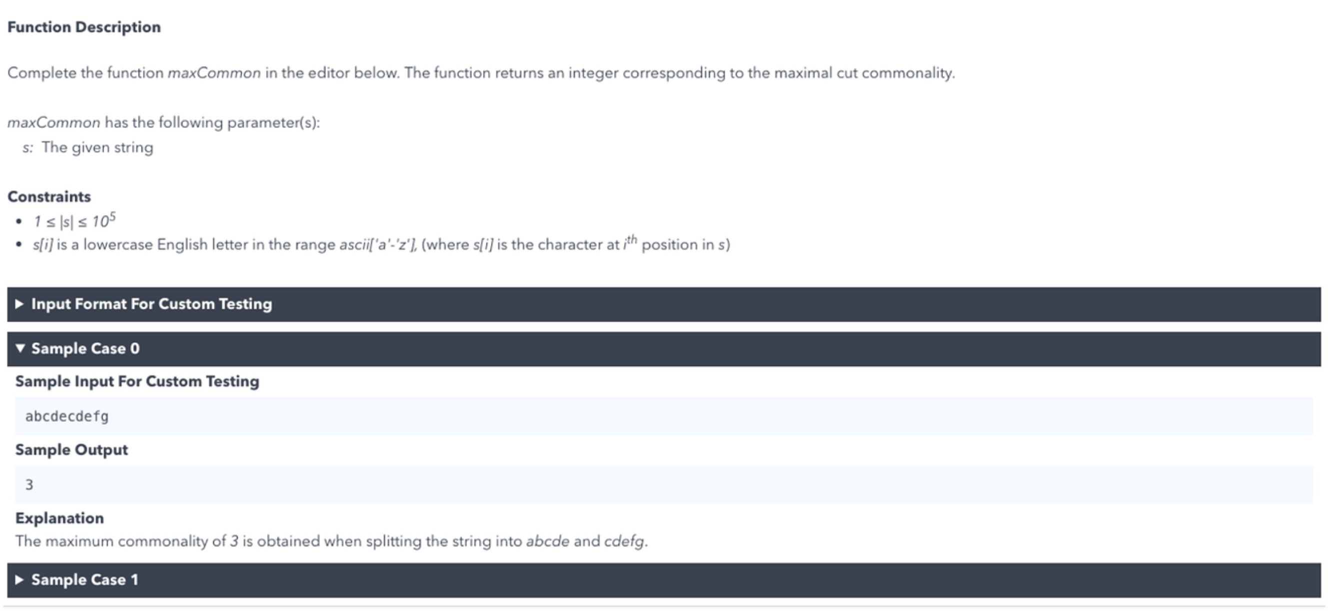
Constraints

- $1 \le n*m \le 10^3$
- $0 \le k \le 10^3$
- *rules[r]* is either "dead" or "alive" (where $0 \le r < 9$)

Solving 1.

```
from copy import deepcopy
def f(grid,k,rules):
    if not grid:
        return grid
    m,n = len(grid),len(grid[0])
    neighbors = [[0,1],[0,-1],[1,0],[1,1],[1,-1],[-1,-1],[-1,0],[-1,1]]
    states = [grid]
    while k > 0:
        k -= 1
        g = [[0 for _ in range(n)] for _ in range(m)]
        for i in range(m):
            for j in range(n):
                count = sum(grid[i][j] for ii,jj in neighbors if 0 <= i+ii
< m and 0 <= j+jj <n)
                g[i][j] = rules[count]
        if g in states:
            i = states.index(g)
            states = states[i:]+states[:i]
            return states[k%len(states)]
        else:
            states.append(g)
            grid = g
    return grid
```

Question 2.



Function Description

Complete the function *maxCommon* in the editor below. The function returns an integer corresponding to the maximal cut commonality.

maxCommon has the following parameter(s):
  s: The given string

Constraints
- $1 \le |s| \le 10^5$
- *s[i]* is a lowercase English letter in the range *ascii['a'-'z']*, (where *s[i]* is the character at i^{th} position in *s*)

▶ **Input Format For Custom Testing**

▼ **Sample Case 0**

Sample Input For Custom Testing

```
abcdecdefg
```

Sample Output

```
3
```

Explanation
The maximum commonality of *3* is obtained when splitting the string into *abcde* and *cdefg*.

▶ **Sample Case 1**

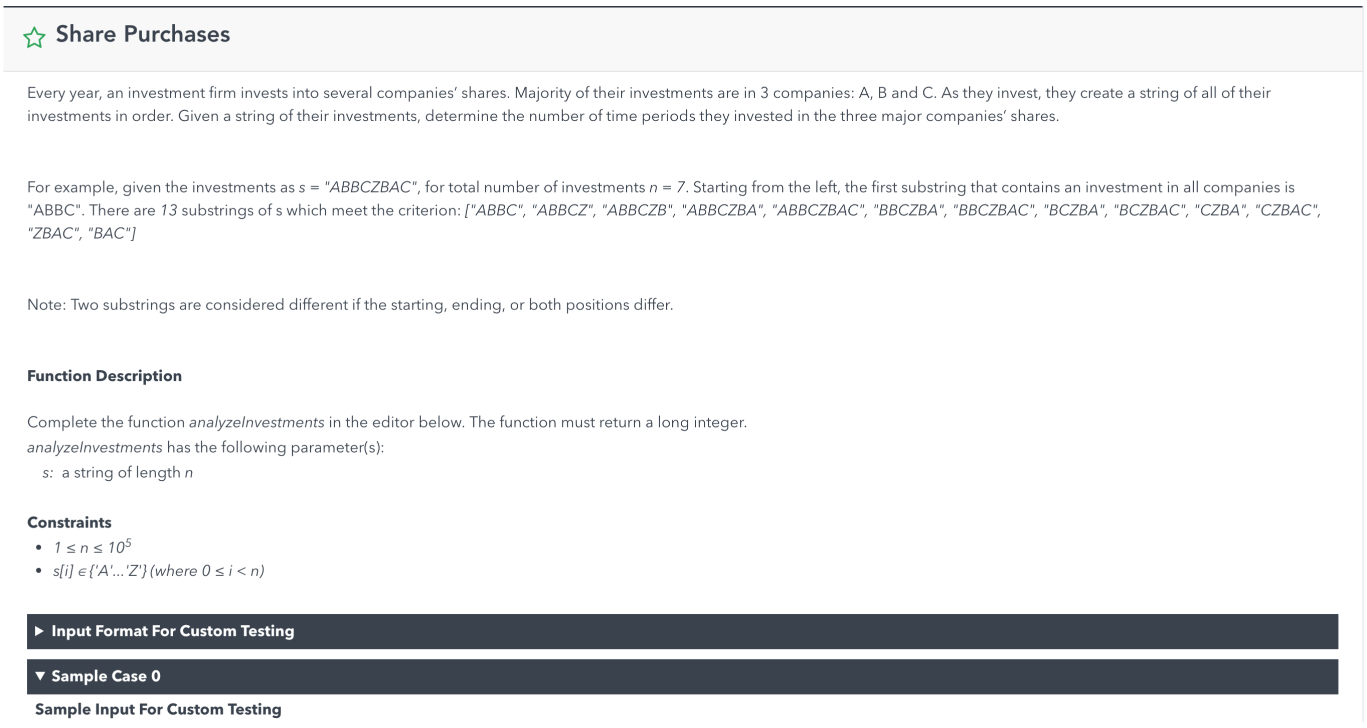
Question 3.

leetcode 1010

Solving 3.

```
def f(times):
    counter = [0]*60
    res = 0
    for i in times:
        i = i % 60
        res += counter[(60-i)%60]
        counter[i]+=1
    return res
```

Question 4.



☆ Share Purchases

Every year, an investment firm invests into several companies' shares. Majority of their investments are in 3 companies: A, B and C. As they invest, they create a string of all of their investments in order. Given a string of their investments, determine the number of time periods they invested in the three major companies' shares.

For example, given the investments as *s = "ABBCZBAC"*, for total number of investments *n = 7*. Starting from the left, the first substring that contains an investment in all companies is "ABBC". There are *13* substrings of s which meet the criterion: *["ABBC", "ABBCZ", "ABBCZB", "ABBCZBA", "ABBCZBAC", "BBCZBA", "BBCZBAC", "BCZBA", "BCZBAC", "CZBA", "CZBAC", "ZBAC", "BAC"]*

Note: Two substrings are considered different if the starting, ending, or both positions differ.

Function Description

Complete the function *analyzeInvestments* in the editor below. The function must return a long integer.
analyzeInvestments has the following parameter(s):
s: a string of length *n*

Constraints
- $1 \le n \le 10^5$
- *s[i] ∈ {'A'...'Z'} (where 0 ≤ i < n)*

▶ Input Format For Custom Testing

▼ Sample Case 0

Sample Input For Custom Testing

Solving 4.

```
def analyzeInvestments(s):
    # Write your code here
    res = 0
    n = len(s)
    if n<3:
        return 0
    abc = {"A":0,"B":0,"C":0}
    left,right = 0,-1
    while right < n and left < n:
        while right < n-1 and not all(abc.values()):
            right+=1
            if s[right] in "ABC":
                abc[s[right]]+=1
        if not all(abc.values()):
            break
        else:
            res+= n-right
            if s[left] in abc:
                abc[s[left]]-=1
            left += 1
    return res
```

Question 5.

> Given the numbers that the program needs to sort and the mapping, i.e. the shuffled version of the decimal digits, return a list of the jumbled numbers sorted by their correct decimal values, ascending. If multiple mapped values are equal, the values returned should be in the original order they were presented.
>
> For example, *mapping = [3,5,4,6,2,7,9,8,0,1]* of fixed length of *m = 10* and another array of numbers strings, *nums = ['990','332','32']* of length *n = 3*.
>
> Map *'990'* as follows:

- The first digit is *'9'*. In the *mapping* array, *9* is at position *6* so the first digit of the mapped value is *'6'*.
- The second digit is *'9'*. Again, the value is at position *6*, so the mapped value is now *'66'*.
- The third digit is *'0'*, found at position *8* of the *mapping* array. The mapped value is *'668'* or *668* as an integer.

> Map *'332'* as:
>
> - The first and second digits are both *'3'* which is found at index *0* in *mapping*. The mapped value is *'00'*.
> - The third digit is *'2'* found at index *4* of *mapping*, so the mapped value is *'004'* or *4* as an integer.
>
> The value *'32'* maps to *'04'*, or integer *4*, which equals the previous value.
>
> Ordering by integer values yields *[4, 4, 668]*, and retaining order for *'332'* and *'32'* results in a return array of associated original values: *['332', '32', '990']*.
>
> **Function Description**
>
> Complete the function *strangeSort* in the editor below. The function must return an array of strings.
>
> *strangeSort* has the following parameter(s):
> *mapping[mapping[0],...mapping[m-1]]*: an array of integers, denoting the first *10* digits in the number system.
> *nums[nums[0],...nums[n-1]]*: an array of strings, denoting the array that needs to be sorted.
>
> **Constraints**
>
> - *m = 10*
> - *$1 \leq n \leq 10^4$*
> - *1 ≤ length of each nums[i] ≤ 100 (where 0 ≤ i < n)*
> - *mapping array will contain all the 10 digits*

Solving 5.

```
def strangeSort(mapping, nums):
    # Write your code here
    maps = [0]*10
    for i,v in enumerate(mapping):
        maps[v] = i
    ans = sorted(int("".join([str(maps[int(i)]) for i in v])),i,v] for i,v
in enumerate(nums))
    return [i[-1] for i in ans]
```