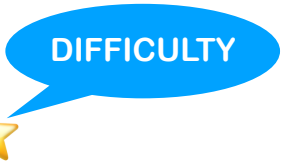
RECURSION

WORKSHEET

CENG 111

2019

GÖKTÜRK ÜÇOLUK



Check if string is palindrome ⭐

Description

Write a function, named **`is_palindrom`**, that takes a string as argument and returns **`True`** if the string is a palindrome, **`False`** otherwise.

Example

```
>>> is_palindrom("BABA")
False

>>> is_palindrom("BABAB")
True

>>> is_palindrom("LEBLEBI")
False

>>> is_palindrom("LEBLEBI")
False

>>> is_palindrom("Rotator")
False

>>> is_palindrom("RotatoR")
True
```

First Recamán's Sequence ⭐

Introduction

The First Recamán's Sequence R is defined as

- R(0) = 0
- R(n) = R(n − 1) − n, if n > 0 ∧ the RHS is positive ∧ R(n) is not already in the sequence,
- R(n) = R(n − 1) + n, otherwise

which can be succinctly defined as "subtract if you can, otherwise add."
The first few terms are 0, 1, 3, 6, 2, 7, 13, 20, 12, 21, 11, ..

Description

Write a function, named **`recaman`**, that takes a positive integer (**`n`**) as argument and returns R(n).

Example

```
>>> recaman(6)
13
```

Note

Care about efficiency !

All possible strings of length k that can be formed from a set of n characters ⭐⭐

Description

Write a function, named **`k_possible`**, that takes a string (**`s`**) and a positive integer (**`k`**) as argument and returns all possible strings of length **`k`** that can be formed from the characters of the string **`s`**.

Example

```
>>> k_possible('ab',3)
['aaa','aab','aba','abb','baa','bab','bba','bbb']

>>> k_possible('abcd',1)
['a','b','c','d']
```

Note

Your ordering of the answer list can be different.

All factorizations ⭐⭐

Description

Write a function, named **`all_factorizations`**, that takes a positive integer (**`n`**) as argument and returns a list all possible factorizations of that number n, where each list holds the factors of a single factorization in ascending order.

Example

```
>>> all_factorizations(16)
[[2,2,2,2],[2,2,4],[2,8],[4,4]]

>>> all_factorizations(12)
[[2,2,3],[2,6],[3,4]]
```

Greatest Common Divisor ⭐

Description

Write a function, named **`gcd`**, that takes two positive integers (**`k,n`**) as argument and returns their greatest common divisor.

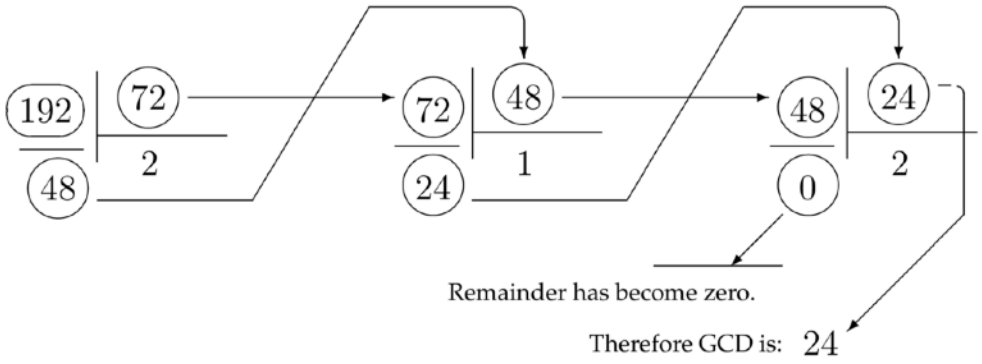
Example

```
>>> gcd(192,72)
24

>>> gcd(72,192)
24

>>> gcd(16,15)
1
```

Hint



Remove duplicates ⭐

Description

Write a function, named **`remove_duplicate`**, that takes a single string argument (**`s`**). The function removes all all but the last of adjacent duplicates.

Example

```
>>> remove_duplicate('geeksforgeeeeek')
'geksforgek'

>>> remove_duplicate('acaaabbbacdddd')
'acabacd'
```

Flatten a list ⭐⭐

Description

Write a function, named **`flatten`**, that takes a single list argument (**`L`**). The elements of **`L`** can be strings, numbers or other lists of such elements. You are expected to remove the nestedness and return a single list.

Example

```
>>> flatten([[1,2,[3,4]],[5,6],7])
[1,2,3,4,5,6,7]


>>> flatten(['ahmet AKHUNLAR',[1930,1998],[['ayse'], \
     'mehmet',['aliye',['yakup','veli']]],[],1.85])
['ahmet AKHUNLAR',1930,1998,'ayse','mehmet','aliye',
 'yakup','veli',1.85]
```

Determinant calculation ⭐⭐⭐⭐

Introduction

In the case of a 2 × 2 matrix the determinant may be defined as

$$|A| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc.$$

Similarly, for a 3 × 3 matrix A, its determinant is

$$\begin{aligned} |A| = \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} &= a \begin{vmatrix} \square & \square & \square \\ \square & e & f \\ \square & h & i \end{vmatrix} - b \begin{vmatrix} \square & \square & \square \\ d & \square & f \\ g & \square & i \end{vmatrix} + c \begin{vmatrix} \square & \square & \square \\ d & e & \square \\ g & h & \square \end{vmatrix} \\ &= a \begin{vmatrix} e & f \\ h & i \end{vmatrix} - b \begin{vmatrix} d & f \\ g & i \end{vmatrix} + c \begin{vmatrix} d & e \\ g & h \end{vmatrix} \\ &= aei + bfg + cdh - ceg - bdi - afh. \end{aligned}$$

Each determinant of a 2 × 2 matrix in this equation is called a minor of the matrix A. This procedure can be extended to give a recursive definition for the determinant of an n × n matrix, the minor expansion formula. So the determinant of an n × n matrix is calculated as a linear combination of (n-1) × (n-1) matrix determinants.

In Python a matrix is usually represented as a list of lists each of which represents a row of the matrix.

Description

Write a function, named **`determinant`**, that takes a single list argument (**`M`**). The elements of **`M`** are equal length lists with floating point elements. You are expected to compute the determinant of the matrix. Your function shall work for all n × n matrices.

Example

```
>>> determinant([[0.4,0,1.2,-4],[7.5,1.1,-0.9,3.9],  \
[11.25,6.8,7.1,12.1],[0,5.7,-0.12,-17.2]]
-2765.76988
```

K-Partition Problem ⭐⭐⭐⭐⭐

Introduction

In k-partition problem, we need to partition an list of positive integers into k disjoint subsets that all have equal sum and they completely covers the set.

Description

Write a function, named **`k_partition`**, that takes a list argument (**`L`**) and a positive integer (**`k`**). If a partition exist, the function returns it as a list of **`k`** many list. If this is not possible the function returns **`None`** as value. If there are more than one solution any of the solutions can be returned.

Example

```
>>> kibele = [7,3,5,12,2,1,5,3,8,4,6,4]
>>> k_partition(kibele,2)
[[5,3,8,4,6,4],[7,3,5,12,2,1]]

>>> k_partition(kibele,3)
[[2,1,3,4,6,4],[7,5,8],[3,5,12]]

>>> k_partition(kibele,4)
[[1,4,6,4],[2,5,8],[12,3],[7,3,5]]

>>> k_partition(kibele,5)
[[2,6,4],[8,4],[3,1,5,3],[12],[7,5]]

>>> k_partition(kibele,7)
None

>>> k_partition([18,2,10],2)
None
```

Note

Your ordering of the answer lists can be different.

n-bit Gray Code ⭐⭐⭐

Introduction

n-bit Gray code is a sequence of all possible n-bit numbers. The property of the Gray Code is that in the sequence successive numbers differ only by one bit. Here are the [2,3,4,5]-bit Gray Codes.

2-bit Gray Code

```
 00 01 11 10
```

3-bit Gray Code

```
 000 001 011 010 110 111 101 100
```

4-bit Gray Code

```
 0000 0001 0011 0010 0110 0111 0101 0100 1100 1101 1111 1110
1010 1011 1001 1000
```

5-bit Gray Code

```
00000 00001 00011 00010 00110 00111 00101 00100 01100 01101
01111 01110 01010 01011 01001 01000 11000 11001 11011 11010
11110 11111 11101 11100 10100 10101 10111 10110 10010 10011
10001 10000
```

Description

Write a function, named **`n_bit_Gray_code`**, that takes a positive integer (**`n`**) as argument and returns the n-bit Gray Code as a list of strings.

Example

```
>>> n_bit_Gray_code(3)
['000','001','011','010','110','111','101','100']
```

Hint

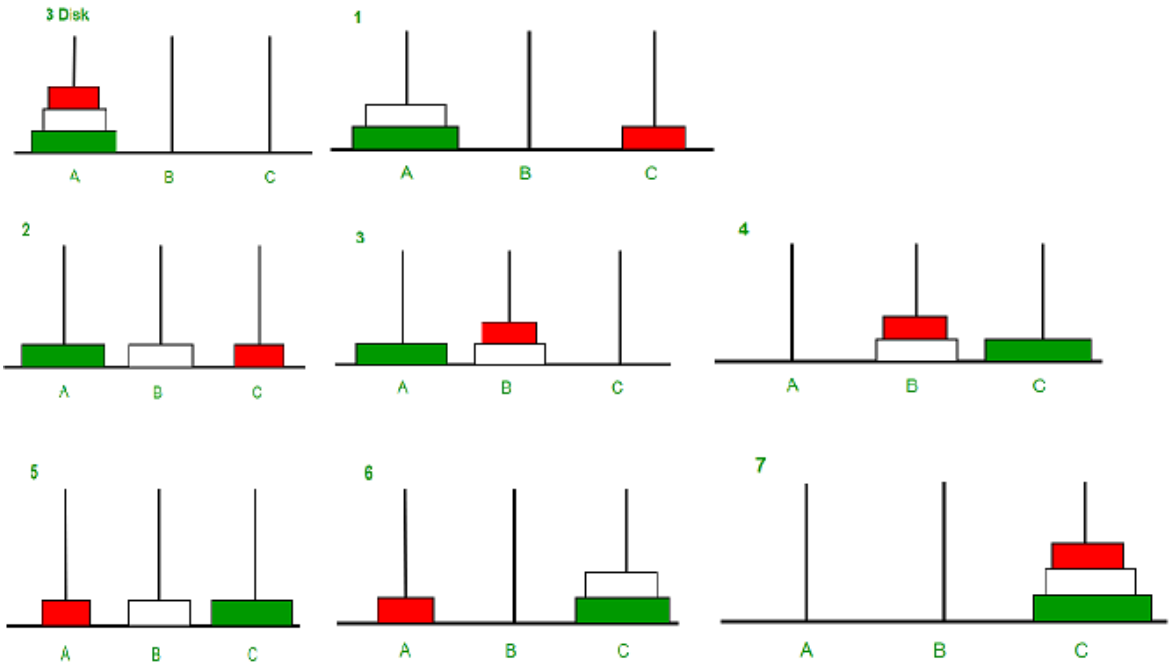
Study the four example sequences. There is a relation between 2 and 3, 3 and 4, 4 and 5 bit Gray Code sequences. Carefully study them and discover the relation. Generalize it to a method to get the (n+1)-bit Gray Code having the n-bit Gray Code to hand.

Tower of Hanoi ⭐⭐⭐

Introduction

Tower of Hanoi is a famous puzzle where we have three rods and N disks. The objective of the puzzle is to move the entire stack to another rod. You are given the number of discs N. Initially, these discs are in the rod A. The objective of the puzzle is to move the entire stack to the rod C, obeying the following simple rules:

1. The discs are arranged such that the top disc is numbered 1 and the bottom-most disc is numbered N.
2. Only one disk can be moved at a time.
3. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
4. No disk may be placed on top of a smaller disk.



Tower of Hanoi solution for N=3

Description

Write a function, named **`TowerOfHanoi`**, that takes a positive integer (**`N`**) as argument and three rod label arguments: (**`from_rod, to_rod, aux_rod`**) and prints out all moves to accomplish the task.

Example

```
>>> TowerOfHanoi(4,'A','C','B')
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 3 from rod A to rod B
Move disk 1 from rod C to rod A
Move disk 2 from rod C to rod B
Move disk 1 from rod A to rod B
Move disk 4 from rod A to rod C
Move disk 1 from rod B to rod C
Move disk 2 from rod B to rod A
Move disk 1 from rod C to rod A
Move disk 3 from rod B to rod C
Move disk 1 from rod A to rod B
Move disk 2 from rod A to rod C
Move disk 1 from rod B to rod C
```

Hint

Believe in ASD.

Coalesce ⭐⭐

Introduction

Some elements of a set may be equivalent. Such equivalences can be expressed using a list of pairs of equivalent elements. Equivalence is symmetric and transitive. Equivalence classes are subsets, all elements of which are equivalent.

Description

Write a function **`coalesce`** which takes a list of pairwise equivalences and return a list of equivalence classes.

Example

```
>>> coalesce([['a','e'],['z','f'],['m','b'], \
['p','k'], ['e','i'],['f','s'],['b','d'],['t','p'], \
['i','o'],['s','v'],['d','g'],['k','p'],['o','u'], \
['v','z'],['g','m'],['p','t']])

[['a','e','i','o','u'],['f','s','v','z'],
['b','d','g','m'],['k','p','t']]
```

Same set ⭐

Description

Write a function **`sameset`** which takes two argument that are lists of any Python data and are representing sets. These lists may contain other lists (to any depth) as elements which are also representing sets. **`sameset`** shall return **`True`** if the two lists are representing the same set and **`False`** otherwise.

Example

```
>>> sameset(['x','a',['b',['c','x',['y','d']],7],[2,3],'u','r'],\
            ['r','a','u',[['x','c',['d','y']],7,'b'],'x',[2,3]])
True

>>> sameset(['x','a',['b',['c',['x','y'],'d']],7],[2,3],'u','r'],\
            ['r','a','u',[['x','c',['d','y']],7,'b'],'x',[2,3]])
False
```