1. Given an N × N chessboard, the goal is to place N queens on the board in such a way that no two queens threaten each other (no two queens share the same row, column, or diagonal).
2. Se solicita a los participantes que implementen una función llamada cambio\_backtracking(denominaciones, total) que tome como entrada una lista de denominaciones de monedas y un valor total para el cambio, y devuelva una lista que represente la cantidad mínima de cada tipo de moneda necesaria para alcanzar el cambio deseado. Se debe utilizar el enfoque de backtracking para resolver este problema.

Aquí hay un ejemplo de cómo debería funcionar la función:

denominaciones = [1, 2, 5, 10, 20, 50, 100, 200]

total = 123

solucion = cambio\_backtracking(denominaciones, total)

print("Solución óptima:", solucion)

El resultado esperado sería:

Solución óptima: [1, 1, 0, 0, 1, 0, 1, 0]

Esto significa que se necesitaría 1 moneda de 1 euro, 1 moneda de 2 euros, 0 monedas de 5 euros, 0 monedas de 10 euros, 1 moneda de 20 euros, 0 monedas de 50 euros, 1 moneda de 100 euros y 0 monedas de 200 euros para alcanzar el cambio de 123 euros.

1. Given a set of items, each with a weight and a value, and a knapsack with limited capacity, the goal is to maximize the total value of the items included in the knapsack without exceeding its capacity.

**Example:**

weights = [2, 3, 4, 5]

values = [3, 4, 5, 6]

capacity = 5

1. You are given a 0-indexed integer array nums of even length consisting of an equal number of positive and negative integers. You should rearrange the elements of nums such that the modified array follows the given conditions:

* Every consecutive pair of integers have opposite signs.
* For all integers with the same sign, the order in which they were present in nums is preserved.
* The rearranged array begins with a positive integer.
* Return the modified array after rearranging the elements to satisfy the aforementioned conditions.

**Example 1:**

Input: nums = [3,1,-2,-5,2,-4]

Output: [3,-2,1,-5,2,-4]

**Explanation:**

The positive integers in nums are [3,1,2]. The negative integers are [-2,-5,-4].

The only possible way to rearrange them such that they satisfy all conditions is [3,-2,1,-5,2,-4].

Other ways such as [1,-2,2,-5,3,-4], [3,1,2,-2,-5,-4], [-2,3,-5,1,-4,2] are incorrect because they do not satisfy one or more conditions.

**Example 2:**

Input: nums = [-1,1]

Output: [1,-1]

**Explanation:**

1 is the only positive integer and -1 the only negative integer in nums.

So nums is rearranged to [1,-1].

1. Given a digit string, return all possible letter combinations that the number could represent. A mapping of digit to letters (just like on the telephone buttons) is given below.



Input:Digit string "23"

Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].

1. Given an integer array nums of unique elements, return all possible subsets (the power set).

The solution set must not contain duplicate subsets. Return the solution in any order.

**Example 1:**

Input: nums = [1,2,3]

Output: [[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]

**Example 2:**

Input: nums = [0]

Output: [[],[0]]

1. An n-bit gray code sequence is a sequence of 2n integers where:

* Every integer is in the inclusive range [0, 2n - 1],
* The first integer is 0,
* An integer appears no more than once in the sequence,
* The binary representation of every pair of adjacent integers differs by exactly one bit,
* The binary representation of the first and last integers differs by exactly one bit.

Given an integer n, return any valid n-bit gray code sequence.

**Example 1:**

Input: n = 2

Output: [0,1,3,2]

Explanation:

The binary representation of [0,1,3,2] is [00,01,11,10].

- 00 and 01 differ by one bit

- 01 and 11 differ by one bit

- 11 and 10 differ by one bit

- 10 and 00 differ by one bit

[0,2,3,1] is also a valid gray code sequence, whose binary representation is[00,10,11,01].

- 00 and 10 differ by one bit

- 10 and 11 differ by one bit

- 11 and 01 differ by one bit

- 01 and 00 differ by one bit

**Example 2:**

Input: n = 1

Output: [0,1]

1. Given a partially filled Sudoku grid, the goal is to fill the grid in such a way that every row, column, and 3x3 subgrid contains all the digits from 1 to 9 without repetition.

**Example**

**Input**

[

[5, 3, 0, 0, 7, 0, 0, 0, 0],

[6, 0, 0, 1, 9, 5, 0, 0, 0],

[0, 9, 8, 0, 0, 0, 0, 6, 0],

[8, 0, 0, 0, 6, 0, 0, 0, 3],

[4, 0, 0, 8, 0, 3, 0, 0, 1],

[7, 0, 0, 0, 2, 0, 0, 0, 6],

[0, 6, 0, 0, 0, 0, 2, 8, 0],

[0, 0, 0, 4, 1, 9, 0, 0, 5],

[0, 0, 0, 0, 8, 0, 0, 7, 9]

]

**Output**

[

[5, 3, 4, 6, 7, 8, 9, 1, 2],

[6, 7, 2, 1, 9, 5, 3, 4, 8],

[1, 9, 8, 3, 4, 2, 5, 6, 7],

[8, 5, 9, 7, 6, 1, 4, 2, 3],

[4, 2, 6, 8, 5, 3, 7, 9, 1],

[7, 1, 3, 9, 2, 4, 8, 5, 6],

[9, 6, 1, 5, 3, 7, 2, 8, 4],

[2, 8, 7, 4, 1, 9, 6, 3, 5],

[3, 4, 5, 2, 8, 6, 1, 7, 9]

]

1. Dado un laberinto representado como una matriz de dimensiones MxN donde cada celda puede ser un pasillo o una pared, determina si existe un camino desde la celda de inicio hasta la celda de salida. Un camino válido se define como una secuencia de celdas adyacentes conectadas por pasillos. Puedes moverte en todas las direcciones (arriba, abajo, izquierda, derecha), pero no a través de las paredes.

**Input**

* Una matriz representando el laberinto donde 1 representa una pared y 0 representa un pasillo.
* Las coordenadas de la celda de inicio y de la celda de salida.

maze = [[1, 0, 0, 0, 0],

[1, 1, 0, 1, 1],

[0, 1, 1, 0, 1],

[0, 0, 1, 1, 1],

[0, 0, 0, 0, 1]

]

start = (0, 0)

end = (4, 4)

**Output**

* True si hay un camino desde la celda de inicio hasta la celda de salida, False de lo contrario.
* Camino a seguir

[ (0, 0), (1, 0), (1, 1), (2, 1), (2, 2), (2, 3), (3, 3), (4, 3), (4, 4)]

1. Dada una matriz de costos costos, donde costos[i][j] representa el costo de asignar la tarea j al trabajador i, implementa una función llamada asignacion\_tareas(costos) que encuentre la asignación óptima de tareas utilizando el algoritmo de backtracking.

**Entrada:**

* costos: Una matriz de tamaño n x n (donde n es el número de trabajadores y también el número de tareas) que contiene los costos de asignar cada tarea a cada trabajador.

**Salida:**

Una lista de tamaño n, donde el elemento en la posición i indica la tarea asignada al trabajador i en la asignación óptima.

El costo total de la asignación óptima.

**Ejemplo:**

# Entrada costos = [ [9, 2, 7, 8], [6, 4, 3, 7], [5, 8, 1, 8], [7, 6, 9, 4] ] # Salida esperada Mejor asignación de tareas: [1, 2, 0, 3] Costo total: 16

**Explicación:**

En este caso, la función asignacion\_tareas encontrará la asignación óptima de tareas y devolverá la lista [1, 2, 0, 3], lo que significa que la tarea 1 es asignada al trabajador 0, la tarea 2 al trabajador 1, la tarea 0 al trabajador 2, y la tarea 3 al trabajador 3. El costo total de esta asignación es 16, que es el costo mínimo posible.