AG1- Actividad Guiada 1

Nombre y Apellidos:
Gloria Angelina Estrada Galindo

Url: https://github.com/Angegloria/03MAIR---Algoritmos-de-Optimizacion---2019/blob/master/AG1/Gloria_Angelina_Estrada_Galindo_AG1.ipynb

• Desarrollar algoritmos de ordenación con python

Los algoritmos de ordenamiento son muy importantes y existen algoritmos de ordenacion lenta y rapida, la diferencia entre estos algoritmos es la eficiencia. El ordenamiento de burbuja es el mas sencillo por eso lo aplique en este caso, que es una lista de numeros no muy grande y por eso es facil ordenarla.

In [1]:
```
mylist = [889,5,4,1,2,8,101,0,20,200,230,119,90,28,390,88,13,1000,396]

def bubble_sort(mylist):
    '''
    Implementacion del algoritmo de burbuja.
    Entrada: my list - objeto list para ser ordenado
    Salida: un objeto list ya ordenado del menor hasta el mayor
    '''
    for i in range(len(mylist)):
        for idx,element in enumerate(mylist):
            if idx == 0:
                continue
            else:
                if mylist[idx]<mylist[idx-1]:
                    element2 = mylist[idx-1]
                    mylist[idx-1] = element
                    mylist[idx] = element2
    return mylist
```

```
bubble_sort(mylist)
```

Out[1]:
```
[0, 1, 2, 4, 5, 8, 13, 20, 28, 88, 90, 101, 119, 200, 230, 390, 396, 88
9, 1000]
```

• Desarrollar algoritmos voraces para resolver problemas

Estos algoritmos se usan para resolver ciertos problemas de optimizacion,y en este caso use un problema simple como el de las monedas, para obtener la solucion al problema de las monedas.Con este algoritmo la idea es que el problema alcance la mejor solución. Otra cosa tambien es que estos algoritmos son muy agresivos y muy eficaces para solucionar algunos problemas, y esta eficacia se usa aun cuando la solución final no sea la mas optima.Por otra parte algunas veces aunque no se tenga la solucion optima, estos algoritmos ayudan a encontrar alguna solucion.

In [2]:
```
import numpy as np
monedas =[50 ,25 ,10, 5 ,1]

def Devolver_Cambio (cantidad):
    cambio = []
    iter_ = 0

    for coin in monedas:

        cambio2 = cambio.copy()
        while sum(cambio2) <= cantidad:
            cambio2.append(coin)
            if sum(cambio2) <= cantidad:
                cambio.append(coin)


    return cambio
```

In [3]:
```
Devolver_Cambio(69)
```

Out[3]:
```
[50, 10, 5, 1, 1, 1, 1]
```

• Desarrollar algoritmos con la técnica de vuelta atrás(backtracking) para resolver problemas.

Los algoritmos de vuelta atras se pueden usar para encontrar soluciones a problemas que tienen determinadas restricciones, la forma de este algoritmo es muy parecida a la de busqueda en profundidad, en este ejercicio use el ejemplo de sudoku.

In [4]:
```
# https://www.codesdope.com/blog/article/solving-sudoku-with-backtracking-c-java-and-python/
import numpy as np
# vuelta atras en la resolucion del sudoku
board = np.array([
    [6,5,0,8,7,3,0,9,0],
    [0,0,3,2,5,0,0,0,8],
    [9,8,0,1,0,4,3,5,7],
    [1,0,5,0,0,0,0,0,0],
    [4,0,0,0,0,0,0,0,2],
    [0,0,0,0,0,0,5,0,3],
    [5,7,8,3,0,1,0,2,6],
    [2,0,0,0,4,8,9,0,0],
    [0,9,0,6,2,5,0,8,1]])

# imprime el tablero en pantalla
def print_board(board):
    for row in board:
        print (row)

# funcion para capturar una celda vacia, devuelve cada vacio en orden
#  tambien va comprobando cada fila
def find_empty(board):
    empty = [-1, -1, False] # inicializa empty con empty[3] = False
    idx_empty = np.argwhere(board == 0)
    if idx_empty.size == 0:
        # Falso significa que ninguna celda está vacía, el tablero está
 completo
        return None
    else:
        return idx_empty
```

```
    return empty

# funcion para comprobar si se puede insertar un valor en una celda
def is_valid(board,number, row, col):
    # row
    if number in board[row,:]:
        return False
    # column
    if number in board[:,col]:
        return False

    # submatrix
    row_grid = (row//3)
    col_grid = (col//3)
    grid_board = board.reshape(3,3,-1,3).swapaxes(1,2)
    if number in grid_board[row_grid][col_grid]:
        return False

    return True

#función para comprobar si podemos poner un
33/5000


#Evalua en una celda particular o no
def solve_sudoku(board):
    row = 0
    col = 0
    #si todas las celdas están asignadas entonces el sudoku ya está res
uelto
    #pasar por referencia porque number_unassigned cambiará los valores
 de fila y columna
    a = find_empty(board)
    if a is None:
        return True

    top_empty = a[0]
    row = top_empty[0]
```

```
    col = top_empty[1]

    for number in range(1,10):
        #si podemos asignar i a la celda o no
        #la célda es tablero[row][col]
        if is_valid(board,number, row, col):
            board[row][col] = number
            #backtracking
            if solve_sudoku(board):
                return True
            #f no se puede proceder con esta solución
            #se reasigna la celda
            board[row][col]=0
    return False

if solve_sudoku(board):
    print_board(board)
else:
    print("No solution")
```

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

• Desarrollar algoritmos con la técnica de divide y vencerás para resolver problemas.

Este algoritmo se usa para solucionar problemas dificiles de forma recursiva al dividirlo en problemas mas pequeños, y de esa forma hacerlo mas simple.Divide y Vencerás tambien es una tecnica de diseño que resuelve un problema a partir de la solución de subproblemas del mismo tipo, pero de menor tamaño. En caso de que los subproblemas sean todavia grandes se aplicara nuevamente la tecnica hasta que alcance subproblemas muy pequeños para solucionarlos de forma directa. Tambien se usa la recursividad en esta tecnica.

In [5]:
```
# se quiere saber el indice del target en una lista ordenada
def binary_search(A,T):
    '''
    Hace una division de la lista en dos partes (binaria) y comprueba s
i el punto medio
    es el objetivo. Si no, entonces descarta la parte que no tiene el o
bjetivo y
    continua dividiendo la otra parte en dos, hasta obtener la respuest
a
    '''

    # limites izquierda (left L) y derecha (right R)
    L = 0
    R = len(A) - 1
    while True:

        if L > R:
            print('La busqueda binaria ha fallado.')
            return 0

        m = int((L+R)/2)
        if A[m] < T:
            L = m + 1
        elif A[m] > T:
            R = m - 1
        else:
            return m
```

In [6]:
```
# Binary search
A = [1, 4, 16, 19,35, 100]
T = 16 # target

binary_search(A,T)
```

Out[6]: 2

• Desarrollar algoritmos con la técnica de programación dinámica para resolver problemas

Este algoritmo de programacion dinamica se usa para solucionar ciertos problemas matematicos seleccionados, donde se toman decisiones en forma secuencial, ademas nos da un proceso sistematico que obtiene las decisiones que maximicen la efectividad al descomponer el problema en etapas, a través de cálculos recursivos. Los algortimos PD se aplican con efectividad a problemas de optimizacion y comienzan de una manera recursiva (ecuacion de Bellman).

In [7]:
```
1>-np.inf
```

Out[7]:
```
True
```

In [8]:
```
# La programacion dinamica para la resolucion del problema de mayor sec
uencia crescente (LIS - longest increasing sequence)
def find_seq(lista):
    n = len(lista)
    max_len = 1
    max_num = -np.inf # inicializa el maximo numero como infinito negat
ivo

    # check guarda los valores de la lista ya vistos
    checked = np.ones(n,dtype=np.int8)*-1

    # length guarda el tamano de la lista LIS en cada paso
    length =  np.zeros(n)
    length[0] = 1


    # hace un loop en cada uno de los elementos
    for i in range(1,n):
        # hace un loop de elemento i de vuelta hasta el inicio de la li
sta
        # y comprueba si el proximo elemento es mayor, entonces increme
nta
        # max_length y pone el indice en la lista checked
        for j in range(i-1,-1,-1):
            if (length[j] + 1) > length[i] and lista[j] < lista[i]:

                length[i] = length[j] + 1
```

```
                checked[i] = j

                if length[i] > max_len:
                    max_length = length[i]
                    max_num = i

    # se usan los indices generados por el algoritmo para escribir la s
ecuencia
    lis = []
    idx = int(max_num)
    # se usan los indices de checked para hacer la lista final (LIS)
    while idx != -1:
        lis.append(lista[idx])
        idx = checked[idx]

    return lis[::-1]
```

In [9]:
```
lista = [100,1,5,0,10,3,200]
find_seq(lista)
```

Out[9]:
```
[1, 5, 10, 200]
```

In [ ]:
```

```