MEMOIZACION

EJERCICIO 1:

1 -Dada una cuadrícula de tamaño m x n, escribí una función que devuelva la cantidad de formas de llegar desde la esquina superior izquierda (0,0) a la inferior derecha (m-1,n-1), moviendo solo hacia la derecha o hacia abajo.

1. def measure\_time(sort\_function, arr):

2.     import time

3.     start\_time = time.time()

4.     sort\_function(arr)

5.     end\_time = time.time()

6.     return end\_time - start\_time

7.

8. def contarCaminosIterativo(m, n):

9.

10.     matriz = [[1] \* n for \_ in range(m)]

11.

12.     for i in range(1, m):

13.         for j in range(1, n):

14.             matriz[i][j] = matriz[i - 1][j] + matriz[i][j - 1]

15.

16.     return matriz[m - 1][n - 1]

17.

18. def contarCaminosMemo(m, n, memo={}):

19.     if m == 1 or n == 1:

20.         return 1

21.

22.     if (m, n) in memo:

23.         return memo[(m, n)]

24.

25.     memo[(m, n)] = contarCaminosMemo(m - 1, n, memo) + contarCaminosMemo(m, n - 1, memo)

26.     return memo[(m, n)]

27.

28. def main\_ejercicio\_1():

29.     m, n = 255, 255

30.     print("EJERCICIO 1: Cantidad de caminos en una cuadrícula (255x255)")

31.     print(f"Tiempo de ejecución (memoizado): {measure\_time(lambda x: contarCaminosMemo(m, n), None)} segundos")

32.     print(f"Tiempo de ejecución (iterativo): {measure\_time(lambda x: contarCaminosIterativo(m, n), None)} segundos")

33.

34.     print(f"Cantidad de caminos (memoizado): {contarCaminosMemo(m, n)}")

35.     print(f"Cantidad de caminos (iterativo): {contarCaminosIterativo(m, n)}")

36.

37.

38.

39. if \_\_name\_\_ == "\_\_main\_\_":

40.     main\_ejercicio\_1()

41.

Texto

El contenido generado por IA puede ser incorrecto.

EJERCICIO 2:

2-Dada una lista de enteros positivos y un número objetivo, escribí una función que determine si existe un subconjunto cuya suma sea exactamente el objetivo.

1. def existeSubconjuntoIterativo(numeros, objetivo):

2.

3.     matriz = [False] \* (objetivo + 1)

4.     matriz[0] = True

5.

6.     for num in numeros:

7.         for i in range(objetivo, num - 1, -1):

8.             matriz[i] = matriz[i] or matriz[i - num]

9.

10.     return matriz[objetivo]

11.

12. def existeSubconjuntoMemo(numeros, objetivo, n=None, memo={}):

13.     if n is None:

14.         n = len(numeros)

15.

16.     if objetivo == 0:

17.         return True

18.     if n == 0:

19.         return False

20.

21.     if (n, objetivo) in memo:

22.         return memo[(n, objetivo)]

23.

24.     if numeros[n - 1] > objetivo:

25.         memo[(n, objetivo)] = existeSubconjuntoMemo(numeros, objetivo, n - 1, memo)

26.     else:

27.         memo[(n, objetivo)] = (existeSubconjuntoMemo(numeros, objetivo - numeros[n - 1], n - 1, memo) or

28.                               existeSubconjuntoMemo(numeros, objetivo, n - 1, memo))

29.

30.     return memo[(n, objetivo)]

31.

32. def main\_ejercicio\_2():

33.     import random

34.     numeros = [random.randint(1, 100) for \_ in range(100)]

35.     objetivo = 1000

36.     print("EJERCICIO 2: Existe subconjunto con suma objetivo")

37.     print(f"Números: {numeros}")

38.     print(f"Objetivo: {objetivo}")

39.     print(f"Tiempo de ejecución (memoizado): {measure\_time(lambda x: existeSubconjuntoMemo(numeros, objetivo), None)} segundos")

40.     print(f"Tiempo de ejecución (iterativo): {measure\_time(lambda x: existeSubconjuntoIterativo(numeros, objetivo), None)} segundos")

41.     print(f"Existe subconjunto (memoizado): {existeSubconjuntoMemo(numeros, objetivo)}")

42.     print(f"Existe subconjunto (iterativo): {existeSubconjuntoIterativo(numeros, objetivo)}")

43.

44. if \_\_name\_\_ == "\_\_main\_\_":

45.     #main\_ejercicio\_1()

46.     main\_ejercicio\_2()

47.

Pantalla de computadora con letras

El contenido generado por IA puede ser incorrecto.

EJERCICIO 3

3-Dado un string s y una lista de palabras válidas, determiná si s se puede formar usando solo palabras de la lista, reutilizándolas si es necesario.

1. def puedeFormarPalabraIterativo(s, palabras):

2.     n = len(s)

3.     matriz = [False] \* (n + 1)

4.     matriz[0] = True

5.

6.     for i in range(1, n + 1):

7.         for palabra in palabras:

8.             if matriz[i - len(palabra)] and s[i - len(palabra):i] == palabra:

9.                 matriz[i] = True

10.                 break

11.

12.     return matriz[n]

13.

14. def puedeFormarPalabraMemo(s, palabras, memo={}):

15.     if s == "":

16.         return True

17.

18.     if s in memo:

19.         return memo[s]

20.

21.     for palabra in palabras:

22.         if s.startswith(palabra):

23.             if puedeFormarPalabraMemo(s[len(palabra):], palabras, memo):

24.                 memo[s] = True

25.                 return True

26.

27.     memo[s] = False

28.     return False

29.

30. def main\_ejercicio\_3():

31.     s = "eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeef"

32.     palabras = ["e"] \* 1000000 + ["f"]

33.     print("EJERCICIO 3: Puede formar palabra")

34.     print(f"String: {s}")

35.     #print(f"Palabras: {palabras}")

36.     print(f"Tiempo de ejecución (memoizado): {measure\_time(lambda x: puedeFormarPalabraMemo(s, palabras), None)} segundos")

37.     print(f"Tiempo de ejecución (iterativo): {measure\_time(lambda x: puedeFormarPalabraIterativo(s, palabras), None)} segundos")

38.     print(f"Puede formar palabra (memoizado): {puedeFormarPalabraMemo(s, palabras)}")

39.     print(f"Puede formar palabra (iterativo): {puedeFormarPalabraIterativo(s, palabras)}")

40.

41. if \_\_name\_\_ == "\_\_main\_\_":

42.     #main\_ejercicio\_1()

43.     #main\_ejercicio\_2()

44.     main\_ejercicio\_3()

45.

Texto

El contenido generado por IA puede ser incorrecto.

EJERCICIO 4:

4-Dado un valor total y una lista de monedas, escribí una función que devuelva cuántas combinaciones distintas de monedas suman ese total.

1. def contarCombinacionesMonedasIterativo(total, monedas):

2.     combinaciones = [0] \* (total + 1)

3.     combinaciones[0] = 1

4.

5.     for moneda in monedas:

6.         for i in range(moneda, total + 1):

7.             combinaciones[i] += combinaciones[i - moneda]

8.

9.     return combinaciones[total]

10.

11. def contarCombinacionesMonedasMemo(total, monedas, n=None, memo={}):

12.     if n is None:

13.         n = len(monedas)

14.

15.     if total == 0:

16.         return 1

17.     if total < 0 or n <= 0:

18.         return 0

19.

20.     if (total, n) in memo:

21.         return memo[(total, n)]

22.

23.     memo[(total, n)] = contarCombinacionesMonedasMemo(total - monedas[n - 1], monedas, n, memo) + \

24.                        contarCombinacionesMonedasMemo(total, monedas, n - 1, memo)

25.

26.     return memo[(total, n)]

27.

28. def main\_ejercicio\_4():

29.     total = 90000

30.     monedas = [1, 5, 10, 25, 50, 100]

31.     print("EJERCICIO 4: Contar combinaciones de monedas")

32.     print(f"Total: {total}")

33.     print(f"Monedas: {monedas}")

34.     print(f"Tiempo de ejecución (memoizado): {measure\_time(lambda x: contarCombinacionesMonedasMemo(total, monedas), None)} segundos")

35.     print(f"Tiempo de ejecución (iterativo): {measure\_time(lambda x: contarCombinacionesMonedasIterativo(total, monedas), None)} segundos")

36.     print(f"Combinaciones (memoizado): {contarCombinacionesMonedasMemo(total, monedas)}")

37.     print(f"Combinaciones (iterativo): {contarCombinacionesMonedasIterativo(total, monedas)}")

38.

39. if \_\_name\_\_ == "\_\_main\_\_":

40.     #main\_ejercicio\_1()

41.     #main\_ejercicio\_2()

42.     #main\_ejercicio\_3()

43.     main\_ejercicio\_4()

44.

Texto

El contenido generado por IA puede ser incorrecto.