DOCUMENTACION PROYECTO 2

CRISTIAN DAVID GARCIA MUÑOZ – 9011001

1. Introducción

Este proyecto tiene como objetivo poner en práctica los conceptos de union y la función de particiones enteras, usando técnicas de recursión con memorización. A través de un simulador, se modelan operaciones mágicas en el mundo ficticio de Setlândia, donde los clanes de espíritus se unen y se analiza cuántas formas hay de distribuir su energía.

2. Objetivo del programa

Simular una secuencia de operaciones de tipo union(x, y) y partitions(x) entre elementos (espíritus) que representan conjuntos disjuntos. A través de estas operaciones, se debe calcular la cantidad de formas posibles de particionar la energía total de un clan (representada por el tamaño del conjunto) como suma de enteros positivos, sin importar el orden, aplicando el resultado módulo 1,000,000,007.

3. Estructura del código

El archivo main.py se compone de las siguientes funciones principales:

leerarchivo(nombre): Esta función tiene como objetivo principal leer el contenido de un archivo de texto y prepararlo para su procesamiento. Al recibir el nombre del archivo como parámetro, abre el archivo en modo lectura y utiliza el método readlines() para obtener todas sus líneas. Cada línea es limpiada mediante strip() para eliminar espacios en blanco y caracteres de nueva línea innecesarios. El resultado final es una lista donde cada elemento corresponde a una línea del archivo original.

```
def leerarchivo (nombre):
    with open (nombre,"r") as file:
        lineas = [linea.strip() for linea in file.readlines()]
    return lineas
```

hallar(grupo, x): Esta función implementa la operación find. Su tarea es identificar el representante (raíz) del conjunto al que pertenece el elemento x. Cuando se invoca, verifica si el elemento es su propio padre (grupo[x] == x); de no ser así, realiza una llamada recursiva para encontrar el verdadero representante y actualiza directamente la referencia. Esta optimización es crucial para mantener la eficiencia del algoritmo, especialmente en operaciones repetitivas, ya que reduce significativamente el tiempo de búsqueda en consultas futuras.

```
def hallar (grupo, x):
    if grupo[x] != x:
        grupo [x] = hallar(grupo,grupo[x])
    return grupo [x]
```

union(grupo, tamaño, x, y): Encargada de combinar dos conjuntos disjuntos, esta función primero determina los representantes de ambos elementos mediante llamadas a hallar. Si los representantes son diferentes, procede a unir los conjuntos aplicando la heurística de unión por tamaño: siempre adjunta el más pequeño al más grande. Además, actualiza el registro de tamaños para reflejar la nueva configuración. La unión por tamaño es una optimización clave que asegura un rendimiento óptimo en operaciones repetidas.

```
def union(grupo,tamaño, x, y):
    conjuntoX = hallar (grupo, x)
    conjuntoY = hallar (grupo, y)

    if conjuntoX != conjuntoY:
        if tamaño[conjuntoX] < tamaño[conjuntoY]:
            grupo [conjuntoX] = conjuntoY
            tamaño[conjuntoY] += tamaño[conjuntoX]
        else:
            grupo [conjuntoY] = conjuntoX
            tamaño [conjuntoX] += tamaño[conjuntoY]
```

particion(z): Diseñada para calcular el número de particiones enteras de un número z, esta función implementa una solución recursiva con memorización. Utiliza un diccionario para almacenar resultados intermedios (memorización), evitando así recalcular los mismos valores múltiples veces. La lógica recursiva considera dos casos principales: incluir el número máximo actual en la partición o excluirlo, sumando ambas posibilidades. Los casos base manejan situaciones donde z es cero (una partición válida) o cuando las condiciones hacen imposible la partición (retornando cero). Cada resultado se devuelve módulo 1,000,000,007 para cumplir con los requisitos del problema y manejar adecuadamente números grandes.

```
def particion (z):
    conjuntoZ = {}

    def resolver (restante, maximo):
        j = (restante, maximo)
        if j in conjuntoZ:
            resultado = conjuntoZ [j]
        elif restante == 0:
            resultado = 1
        elif restante < 0 or maximo == 0:
            resultado = 0
        else:
            forma1 = resolver (restante - maximo, maximo)
            forma2 = resolver (restante, maximo -1)
            resultado = (forma1 + forma2) % 1000000007

        conjuntoZ [j]= resultado
        return resultado
    return resolver (z, z)
```

ejecutar(nombreAch): Actuando como el núcleo del programa, esta función orquesta todo el proceso. Comienza leyendo el archivo de entrada mediante leerarchivo, luego procesa cada caso de prueba secuencialmente. Para cada caso, inicializa grupo y tamaño, que representan los conjuntos disjuntos y sus respectivas magnitudes. A medida que procesa cada operación, distingue entre uniones (que modifican la estructura de conjuntos) y consultas de particiones (que generan resultados). Implementa validaciones para asegurar que los parámetros de entrada cumplan con las restricciones establecidas. Finalmente, recolecta e imprime los resultados de todas las operaciones partitions en el orden requerido, cumpliendo así con la salida esperada del programa.

```
def ejecutar (nombreAch):
    lineas = leerarchivo(nombreAch)
    indice = 0
    Ncasos = int(lineas[indice])
    indice += 1

    if not (1<=Ncasos<=10):
        print ("el limite debe ser entre 1 y 10")
        return

    resultados = []

    for i in range(Ncasos):
        partes = lineas[indice].split ()
        n = int(partes[0])
        m = int(partes[1])
        indice += 1

        if not (1<= n <=50) or not (1<= m <=100):
            print ("el limite de n debe ser entre 1 y 50 y el de m debe ser entre 1 y 100")
            return

        grupo = [j for j in range(n + 1)]
        tamaño = [1] * (n + 1)

        for i in range (m):
            operaciones = lineas[indice].split ()
            indice += 1

            if operaciones[0] == "union":
                x = int(operaciones[1])
                y = int(operaciones[2])
                union (grupo, tamaño, x, y)
            elif operaciones[0] == "partitions":
                x = int(operaciones[1])
                resultado = particion (tamaño[hallar(grupo, x)])
                resultados.append(resultado)

    for l in resultados:
        print (l)

ejecutar("a.txt")
```