class DisjointSet - Unión y búsqueda

Clase que gestiona conjuntos disjuntos:

- self.padre= list(range(n + 1): Cada elemento es su propio padre al iniciar. Por ejemplo si n = 3, el padre seria [0, 1, 2, 3]
- self.tamaño = [1] (n + 1): El tamaño de cada conjunto empieza en uno

- Funciones:
  - find(self, x): Busca la raíz del conjunto al que pertenece.
    - Primero revisa si x no es su propio padre, lo que confirma que x no es la raíz de su conjunto. Luego, se busca recursivamente el padre raíz y por último se retorna.
  - union(self, x, y): Une los conjuntos de “x” y “y“ por tamaño.
    - Se obtiene la raíz del conjunto de x y y con la función find(). Después, se verifica que sean conjuntos distintos. El árbol más pequeño se une al más grande y se suman los tamaños. Ejemplo si el tamaño de la raiz_x es más pequeño, lo une a raíz_y y luego al tamaño de raíz_y, se le suma el tamaño de la raiz_x.
  - size(self, x): Devuelve el tamaño del conjunto al que pertenece x.
    - Se encuentra la raíz usando find(), y retorna el valor que está en el arreglo tamaño para esa raíz.

contar_particiones(n)

Esta Función usa memorización para contar el número de formas en que un número n puede ser expresado como suma de enteros positivos.

Se crea una variable llamada resultado y se inicializa en 0. Luego, se verifica si n ya fue calculado, si es así, devuelve el valor que se calculó anteriormente. Si ese valor no se ha calculado, entonces se hace una función anidada llamada contar(restante, máximo).

En esta función contar(restante, máximo), donde restante es el número que queremos particionar, es decir lo que "va quedando" por descomponer y máximo es el valor máximo que se permite para los sumandos en la partición actual.

En esta función de contar(), se crea una variable llamada resultado_local, si restante == 0, entonces el resultado_local pasa a ser 1 porque hay 1 forma de particionar 0. Otra

opción es si el restante es menor que 0 o máximo es igual a 0, en esos casos, resultado_local pasa a ser 0 porque no hay particiones validas. Si no se cumplen alguna de estas condiciones, entonces creo otra variable que es combinación = (restante, máximo), y se vuelve a mirar, si ya se calculo la clase entonces resultado_local es el valor ya calculado. Si no se ha calculado todavía, hago una variable llamada suma que usa la recursividad, la cual es la siguiente:

Parte 1 + Parte 2

```
suma = (contar(restante – maximo, maximo) + contar(restante, maximo – 1)) % MOD
```

```
f (n, k) =          f (n – k, k)          +        f (n, k – 1)
```

1. **Parte 1 – incluyendo al máximo:**
    - Se resta máximo a restante (porque ya lo usamos como termino).
    - Se sigue permitiendo máximo en las siguientes particiones.
    - Ejemplo*:* Para particionar 5 con máximo 3, calculamos contar(2,3), porque 5 - 3 = 2.
2. **Parte 2 – excluyendo al máximo:**
    - El restante no cambia (no hemos usado máximo).
    - Se reduce el máximo a máximo - 1 para los siguientes términos.
    - Ejemplo: Para particionar 5 con máximo 3, calculamos contar(5,2).

Al final, se suman los dos resultados y se aplica % MOD (10**9 + 7) para evitar desbordamientos como se pide en el proyecto.

Cada vez que se hace esa operación el resultado lo guardo en el diccionario particiones_mem con key “combinación” para la memorización. Y al final el resultado_local pasa a ser la suma que se calculó.

procesar_caso(n, m, operaciones)

Esta función procesa un caso de prueba con n elementos y m operaciones. Las operaciones pueden ser:
- "union x y”: Unir elementos x y y.

- "partitions x": Calcular cuántas particiones posibles tiene el tamaño del conjunto de x. Devuelve una lista de resultados para cada operación "partitions".

Primero creo una clase de conjuntos disjuntos con n elementos. Creo una lista llamada resultado para agregarlos. Luego, hago un ciclo por cada elemento que haya en la lista operaciones, y la convierto a una lista de strings, para separar por partes el mensaje que entra el usuario, el cual la primera parte, es decir [0], nos va a decir si es union o partitions. Si [0] es union x y y, paso a enteros la parte[1] y parte[2] respectivamente, luego ejecuta la operación unión, que se definió en la estructura de conjuntos disjuntos. En cambio, si [0] es partitions, se pasa a entero la parte[1], luego se crea una variable llamada tam en el cual se obtiene el tamaño del conjunto al que pertenece x usando el método size de la clase DisjointSet. (Cabe aclarar que cuando genero una lista llamada partes para cada operación, es una lista muy pequeña, si es unión contiene 3 elementos, y si es partición contiene 2 elementos. Además, son temporales, es decir, estas listas se eliminan después de cada ciclo, por lo que no afecta mucho en la parte del uso de la memoria).

Después, en la variable resultado se calcula el número de particiones del conjunto de tamaño tam usando la función contar_particiones anteriormente definida. Convierto la variable resultado a string y lo agrego a la lista llamada resultados. Por último, retorno la lista de resultados.

main()

En esta función, primero se lee el número de casos que se desean hacer y lo guardo en una variable llamada “T” (lo hice con input ya que es solo un número que se ingresa una vez). Luego, hago un ciclo para que se repita “T” veces; en ese ciclo leo “n” y “m” que son números que están separados por un espacio, por lo que uso stdin.readline (porque es una línea que tiene varios argumentos) también .strip() .split() para eliminar los espacios y para separarlos en una lista de strings. Después, tomo el número en la posición 0 de la lista n_m y lo convierto a un entero para guardarlo en la variable “n”, que es la cantidad de elementos. Luego tomo el número en la posición 1 y lo guardo en la variable “m”, que representa la cantidad de operaciones a realizar. Y creo una lista vacía llamada operaciones. Hago otro ciclo para que se repita “m” veces. Esta vez, leo “op” (que es la operación ya sea unión o partición) y lo agrego a la lista de operaciones.

Declaro la variable resultados sea el llamado a la función procesar_caso() pasándole la cantidad de elementos “n”, el número de operaciones “m” y la lista de operaciones. Esta

función procesa todo y devuelve una lista con los resultados de las operaciones tipo partitions. Por último, hago recorrido por cada resultado y lo imprimo.