Jaime Andres Torres Bermejo
202014866
andrestbermejoj@gmail.com

Proyecto 1
Diseño y Análisis de Algoritmos

2023-03-31

1 Enunciado del Proyecto

Uno de los principales retos de la reforma a la salud 2023 propuesta por el gobierno actual son los denominados Centros de Atención Primaria (CAD). A cada CAD en el país serán asignados los beneficiarios y sus familias según cercanía y disponibilidad. Para que las finanzas de estos centros no se vean afectadas por una mala distribución de beneficiarios y para garantizar el mejor servicio, es necesario diseñar una estrategia de asignación equitativa de beneficiarios.

Problema: Suponga que para un atender la demanda de salud en un determinado municipio de Colombia se habilitaran *k* diferentes CADs a los cuales se deberán asignar *m* familias diferentes. La familia i- esima cuenta con f_i miembros diferentes. Se le encarga la tarea de identificar si existe una manera en que todos los CAD puedan quedar con la misma cantidad de beneficiarios, con la restricción que TODOS los miembros de un mismo grupo familiar sean asignados a un mismo CAD. [1]

2 Especificaciones del proyecto

2.1 Entorno de Desarrollo y entorno de prueba.

La solución propuesta fue probada en el siguiente entorno de Desarrollo, debería funcionar en computadores que compartan las dependencias principales, como Python o un sistema operativo basado en Unix, así no comparta necesariamente el hardware. No se proporciona un entorno de virtualización por lo que recomiendo intentar correrlo en un entorno relativamente similar.

- **Sistema Operativo:** Ubuntu 22.10 x86_64, Minimal Install. También probado con Manjaro Linux 22.0 x86_64
- **Kernel:** Linux 5.19.0-38-generic
- **Lenguaje de Programación:** Python 3.10.7 x86_64, Compilado desde apt como un .deb, incluye python3-pip, También probado con la versión de Arch
- **Librerías Utilizadas**
  - os
- **Procesador:** AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx (8) @ 2.100GHz
- **IDE:** Visual Studio Code 1.76.2, instalado desde un paquete snapd. Incluye extensiones de Git y Python. En Manjaro fue instalado desde el AUR con el paquete visual-studio-code-bin

2.2 Estructura del Input y Output

A primera vista, este pareciese ser un problema bastante extraño, si fuesemos a tomarlo por la lógica del propio problema sin necesariamente cuestionarla. Sin embargo, esta estructura puede verse de forma mucho más clara al ver la forma como un input esta estructurado, y a partir de esto, comprobar la utilidad de cada parte del input. Entonces, tomemos como ejemplo el segundo test proveido por el enunciado:

```
3
3 5 8 7 3 4 11
2 3 4 7 9
2 4 5 7 8 4
```

Y de este input, tendríamos este output:

```
True [(8,3),(7,4),(11)]
False
True [(5,7)(8,4)]
```

Para entender el porque de este output, vamos a tomar el primer input de este set de inputs:

```
3 5 8 7 3 4 11
True [(8,3),(7,4),(11)]
```

La primera cosa a notar es que este input no tiene toda la lista que se pasa por parámetro. Los dos primeros numeros de la lista se obvian porque **No representan familias, sino otras variables que nos serán relevantes.** Especificamente, el primer valor representa las *k* instancias de CAD, que serán listas, y el segundo valor representa las *i* familias que existen dentro del sistema, cuyo numero de miembros es el int que representan. Por lo tanto, el input se vería, una vez formateado, así:

$$Input = \begin{cases} i = 5 \\ k = 3 \\ nums = [8, 7, 3, 4, 11] \end{cases}$$

notese que *i* coincide con la longitud de *nums*, y *k* coincide con la cantidad de listas generadas en el output. Esto no es una coincidencia, y será muy importante cuando hablemos del diseño de la solución, pero por ahora, limitemonos a pensar en que esto cambia la forma en la que pensamos del concepto de CADS y familias, a listas Y enteros, entonces cuando pensamos el problema, no nos centramos en el enunciado, sino en su abstracción, **si encontramos las *k* tuplas de los valores de nums con suma igual, resolveremos el problema.**
Para corroborar esto, vamos a tomar la lista del output y vamos a ordenarla

$$output[1] = [(8, 3), (7, 4), (11)]$$
$$< suma(Aritmetica) >$$
$$output[1] = [[\underbrace{3 + 8}_{11}], [\underbrace{7 + 4}_{11}], [\underbrace{11}_{11}]]$$

Como es posible tomar la lista *nums* y ponerla en *k* tuplas de sumas exactamente iguales, entonces el resultado es True, si no fuese posible, simplemente se retornaría False, la estructura dinámica de Python y sus retornos nos permitirá hacer de forma bastante sencilla la solución, pero nos estamos adelantando.

2.3 Lectura de datos

Según se nos fue informado por los monitores, la lectura se hará a partir de la lectura de archivos de texto y, por lo tanto, la estructura del archivo de Python esta pensada para poder leer varios archivos de texto secuencialmente. Para esto hemos importado la librería 'os' de las librerías base de Python, con esta y el

3 Diseño de la Solución

3.1 Conceptos Teóricos Importantes

3.1.1 Programación Dinámica

3.1.2 Memorización

La Memorización se define cómo: 'a'[2]

3.2 Conceptos Prácticos Importantes

3.2.1 Tipado dinámico

El lenguaje de Programación utilizado para resolver este algoritmo fue Python, el cuál tiene la capacidad de mantener una estructura de tipado dinámica, esta implica que podemos declarar una variable de un tipo y esta podrá ser interpretada como de otro tipo a la hora de compilar el código, otros lenguajes de Programación con esta caráccteristica son Ruby y Javascript

3.2.2 Return vacío

Python nos permite hacer un return nulo, esto puede ser utilizado para imprimir el resultado de una operacion sin necesidad de preocuparnos por el retorno de este en un entorno recursivo.

4 Análisis de la Solución

4.1 Problemas similares

4.1.1 Leetcode 416. Partition Equal Subset Sum

El problema de este proyecto esta pensado como, en realidad, una generalización del problema 416 de Leetcode, Partition Equal Subset Sum[3], el cual se lee de la siguiente manera:

```
Given an integer array nums, return true if you can partition the array
into two subsets such that the sum of the elements in both subsets is
equal or false otherwise.

//Example 1
Input: nums = [1,5,11,5]
Output: true
Explanation: The array can be partitioned as [1, 5, 5] and [11].

//Example 2
Input: nums = [1,2,3,5]
Output: false
Explanation: The array cannot be partitioned into equal sum subsets.
```

El problema resuelto en este Proyecto puede considerarse como una generalización en K subarreglos de este problema, y por eso mismo nos concierne la estructura de esta solución para poder entender la forma como se pudo generalizar de 2 subarreglos a *k* subarreglos.

Estrategia de solución Al leer las soluciones propuestas del problema, nos vamos a dar cuenta de dos técnicas ampliamente usadas: Top-Down y Memorización. Estas

Diferencias con el problema acá planteado Dado que estamos generalizando el problema a la hora de hablar de i familias en k CADs, los casos base propuestos en este problema no pueden ser aplicados, ya que de por si mismos no nos garantizan la correctitud del problema. Por ejemplo, pensemos en el caso base:

```
//si la suma total de nums es impar, retorna false  de forma automática
if(total_sum%2): return False
```

Este caso no nos va a servir dentro de la lógica de nuestra solución porque el valor de k puede ser mayor a 2, y por esto, pueden forzarse situaciones en las que la suma total sea impar y el resultado sea 'True', con el fin de entender esto, considerese este contraejemplo a partir del test 1:

$$Lemma1 : (totalSum\%2) = False$$

$$Lemma2 : [8, 7, 3, 4, 11] \rightarrow True \wedge [(8, 3), (7, 4), (11)]$$

$$< Lemma1,\ True \wedge q = True,\ Lemma2,\ 0 = Boolean\ False >$$

$$True \rightarrow totalSum([8, 7, 3, 4, 11])\%2 = 0$$

$$< totalSum([8, 7, 3, 4, 11]) >$$

$$True \rightarrow 33\%2 = 0$$

$$< 33\%2 = 1,\ 1 = Boolean\ True,\ Lemma2 >$$

$$True \rightarrow False$$

$$< True\ can't\ imply\ False >$$

$$False$$

Por lo tanto, no podemos esperar que todos los casos sean divisibles por 2, y aunque una solución de este Leetcode nos podría resolver una cantidad limitada de casos, no podemos subsistir de su lógica completamente con el fin de resolver este problema, pues causará falsos negativos en casos donde la suma sea impar.

4.2 Complejidad

4.3 Limitantes

- el archivo solo puede recibir inputs de archivos de texto, un input no puede ser ingresado directamente desde el usuario a partir de la consola
- es posible que el programa muera intentando analizar subcarpetas de su carpeta de ingreso, por lo que se recomienda mantener los archivos de prueba en el entorno raíz de la carpeta 'tests'.
- Con el fin de que se pueda abrir en diferentes computadores usamos una ruta relativa a la hora de escribir el entorno de ficheros de donde tomará las pruebas, si se usa este para calificar, es necesario abrir en el entorno desde la primera carpeta del proyecto en caso de usar un IDE, se puede modificar la ruta de lectura, caso en el cuál debería poderse.

4.4 Consideraciones

- La capacidad de tipado dinámico de Python es directamente responsable de la capacidad de este algoritmo en algunos casos, gran parte de las funciones de carácter lambda deberían reescribirse en caso de que se fuese a reescribir la solución en un sistema de tipado estático.
- se omite la primera linea del archivo que leemos porque se asume tiene la cantidad de casos de prueba necesarios para el archivo, esta información nos sería relevante en lenguajes como C, donde se debe configurar manualmente el manejo de ciclos en situaciones como esta.
- A día de hoy, este es un trabajo en grupo que realicé solo, por favor ten eso en cuenta, profe.

5 Referencias.

1 : *Enunciado del proyecto, ISIS 1105 Diseño y Análisis de Algoritmos Semestre 2023-10. Proyecto – PARTE 1* Uniandes, 2023.

2 : *The Algorithm Design Manual, Springer, Second Edition*, Steven S. Skiena, 2008.

3 : *416. Partition Equal Subset Sum*, Leetcode, recuperado de https://leetcode.com/problems/partition-equal-subset-sum/ , 2023.