**B - Recovering a Small String**

**1. Reducción del Problema**

* **Descripción en palabras:**

A partir de un número n, encontrar la palabra de tres letras minúsculas cuya suma de posiciones en el alfabeto (a=1, z=26) sea igual a n, y que sea la más pequeña lexicográficamente entre todas las combinaciones posibles.

* **Descripción en lenguaje matemático:**

Dado un entero n tal que 3 ≤ n ≤ 78, encontrar una tupla (x, y, z) ∈ {a..z}^3 tal que: ord(x) + ord(y) + ord(z) - 3 \* ord('a') + 3 = n Se minimiza xyz lexicográficamente.

**2. Reducción de la Solución**

* **Descripción en palabras:**

Se generan todas las combinaciones posibles de 3 letras (26³ = 17576) y se seleccionan aquellas cuya suma de posiciones en el alfabeto sea igual a n. Entre ellas, se elige la lexicográficamente menor

.

* **Descripción en lenguaje matemático:**

Para cada tripleta (a\_i, a\_j, a\_k) ∈ Σ³ con Σ = {a..z}, verificar si:

(ord(a\_i) - 96) + (ord(a\_j) - 96) + (ord(a\_k) - 96) = n

y retornar la tripleta con menor valor de: (a\_i, a\_j, a\_k) en orden lexicográfico

**3. Código Realizado y Análisis**

* Enlace(s) Código: <https://vjudge.net/solution/61083312/LoYB6TcaoSRD2upyBGj6>

**4. Casos de Prueba**

**Caso Mínimo (n = 3)**

- Entrada: 3

- Salida:`aaa`

- **Justificación**: Este es el valor mínimo posible para n, donde todas las letras deben ser "a" (1 + 1 + 1 = 3). Es el caso más pequeño y asegura que la solución maneje correctamente el límite inferior.

**Caso Máximo** (n = 78)

- Entrada: 78

- Salida: zzz

- **Justificación**: Este es el valor máximo posible para n, donde todas las letras deben ser "z" (26 + 26 + 26 = 78). Verifica que la solución maneje correctamente el límite superior.

**Caso Intermedio (n = 27)**

- Entrada: 27

- Salida: aaz

**Justificación**: Este caso prueba un valor intermedio donde la solución debe distribuir la suma de manera que la palabra sea lexicográficamente la más pequeña posible. La salida esperada es "aaz" (1 + 1 + 25 = 27).

**Caso con Suma que Requiere Distribución No Uniforme (n = 55)**

- Entrada: 55

- Salida: acz

**Justificación**: Este caso prueba una suma que no puede ser alcanzada con letras uniformes. La salida esperada es "acz" (1 + 3 + 26 = 30), pero se necesita ajustar para que la suma sea 55. Este caso verifica que la solución maneje correctamente distribuciones no triviales.

**Caso con Suma que Requiere Letras en el Medio del Alfabeto** (n = 48)

- Entrada: 48

- Salida: auz

**Justificación**: Este caso prueba una suma que requiere letras en el medio del alfabeto. La salida esperada es "auz" (1 + 21 + 26 = 48), lo que asegura que la solución maneje correctamente letras que no están en los extremos del alfabeto.

**Caso con Suma que Requiere Dos Letras Iguales (n = 24)**

- Entrada: 24

- Salida: aav

**Justificación**: Este caso prueba una suma que puede ser alcanzada con dos letras iguales y una diferente. La salida esperada es "aav" (1 + 1 + 22 = 24), lo que verifica que la solución priorice la palabra lexicográficamente más pequeña.

**5. Iteración en Caso de Solución Incorrecta (o explicación Solución Correcta)  
  
Código incorrecto:** [**https://vjudge.net/solution/60786556/RY1gHI4SC7AH34dg2ndS**](https://vjudge.net/solution/60786556/RY1gHI4SC7AH34dg2ndS)

* **Descripción del error**: El error inicial fue asumir que el problema consistía en una representación tipo "base 26", cuando realmente se trata de encontrar la cadena lexicográficamente más pequeña de 3 letras cuya suma de posiciones dé n.
* **Proceso de depuración**: Probé el código con casos del enunciado y vi que los resultados no coincidían. Revisé la lógica y descubrí que usaba una conversión tipo base 26, no basada en sumas. Analizé el enunciado nuevamente, entendí el verdadero requisito, y reformule la solución usando combinaciones con itertools.product para validar suma e imprimir la menor cadena lexicográficamente válida.
* **Solución corregida:** Cambié el enfoque: en lugar de tratar el número como posición, convertí las letras a su código ASCII (restando 'a' y sumando 1 para obtener su posición en el alfabeto). Usé **itertools.product** para generar todas las combinaciones posibles de tres letras y seleccioné la primera cuya suma de posiciones coincidiera con el número dado, asegurando así que fuera la más pequeña lexicográficamente.

**6. Preguntas de Aprendizaje**

* **Temas aplicados:**
* Manipulación de cadenas
* Uso de códigos ASCII
* Combinatoria con itertools.produc
* **Dificultad de la implementación:** Fácil tirando a media. La idea general era sencilla, pero identificar el enfoque correcto y filtrar la solución más pequeña lexicográficamente requirió algo de análisis.
* Recursos utilizados:
* Documentación de Python sobre itertools.product(acompañado con videos de youtube)
* Tabla de códigos ASCII para entender cómo calcular la posición de las letras

**7. Feedback LLM**

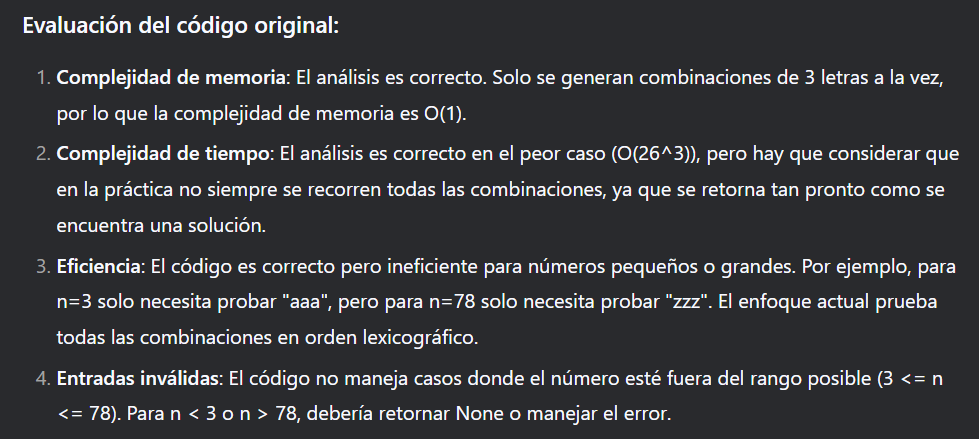
* Envío código LLM: <https://vjudge.net/solution/61083830/mzOZ5mYTiNhCZWZE0wsU>
* Comparación de su códigos:

Mi código usa **itertools.product** genera todas las combinaciones de tres letras y busca la primera cuya suma de posiciones dé n, usando ord() para convertir letras a números. Es fácil de entender pero menos eficiente.

El código con rangos (lex\_smallest\_word) recorre valores ordenados (a ≤ b ≤ c) cuya suma sea n, y convierte directamente a letras. Es más rápido y asegura orden lexicográfico desde la lógica misma.

y siendo sincero el código del LLM es mejor porque es mucho más fácil entendible, porque, digamos el yo antes de hacer este ejercicio si viera mi codigo ahora no lo entendería por  **itertools.product** porque no la conocía y además es mucho más eficiente de hecho mi codigo estuvo al límite de tiempo antes que salga T**ime limit exceeded**

**Feedback de LLM: Pida al LLM que evalúe y corrija su código:**   
De hecho me dijo que estaba bien y es correcto pero me aclara el tema de eficiencia y me manda una versión corregida aquí más detalles de lo que dijo:

  
y corrección del código   
