Análisis de problema 3.

Este problema se puede abordar como un caso de programación dinámica, donde el objetivo es minimizar la cantidad de vitaminas necesarias para alcanzar un límite dado en el aumento de velocidad de un Pokémon. Cada vitamina tiene un valor asociado que indica cuánto aumenta la velocidad.

En nuestro problema los objetos son las vitaminas, el peso de cada objeto es el aumento de velocidad que proporciona, el valor de cada objeto es 1, y el límite de peso es el límite de aumento de velocidad. El algoritmo aplicado es una solución óptima para el Knapsack Problem. La complejidad temporal del algoritmo es O(G * V), donde G es el límite de aumento de velocidad y V es el número de vitaminas.

Pseudocodigo:

```
Función minVitamins(G, vitamins, V):
   // Inicializar arreglo para almacenar la cantidad minima de vitaminas para cada valor de velocidad.
   dp[G + 1]
   Para i en rango (0, G):
      dp[i] = 1000000 // Usar un valor grande para representar "infinito".
   dp[0] = 0 // No se necesitan vitaminas para un aumento de 0.

   // Recorrer todas las vitaminas.
   Para i en rango (0, V):
      Para j en rango (vitamins[i], G+1):
         // Actualizar el valor de dp[j] si es posible alcanzarlo con menos vitaminas.
         dp[j] = min(dp[j], dp[j - vitamins[i]] + 1)

   // Devuelve la cantidad minima de vitaminas para alcanzar G o -1 si no es posible.
   Si dp[G] == 1000000:
      Devolver -1
   Sino:
      Devolver dp[G]

// En el programa principal:
Leer el número de casos de prueba T
Mientras T sea mayor que 0:
   Leer G y V
   Leer la lista de valores de vitaminas
   Calcular y mostrar el resultado usando minVitamins
   Restar 1 a T
```

Ejecución manual:

G = 3 (límite de aumento de velocidad)

V = 2 (número de tipos de vitaminas)

vitamins = [1, 2] (valores de aumento de velocidad de cada vitamina)

Inicialización de dp:

dp[0] = 0

dp[1] = 1

dp[2] = 1

dp[3] = 1000000

Después de la iteración para la vitamina 1:

dp[1] = 1

dp[2] = 1

dp[3] = 2

Después de la iteración para la vitamina 2:

dp[2] = 1

dp[3] = 2

Resultado Final:

dp[3] = 2

Por lo tanto, la cantidad mínima de vitaminas necesarias para alcanzar un aumento de velocidad de 3, con las vitaminas disponibles [1, 2], es 2.

Explicación del código elaborado:

Utilizamos programación dinámica (Knapsack Problem) para calcular la cantidad mínima de vitaminas necesarias para alcanzar un límite de aumento de velocidad (G), considerando varios tipos de vitaminas con sus respectivos aumentos de velocidad.

La función *minVitamins* utiliza un enfoque de programación dinámica para calcular la cantidad mínima de vitaminas necesarias para cada nivel de aumento de velocidad desde 0 hasta G. Se crea un arreglo *dp* para almacenar los resultados intermedios. Itera sobre cada tipo de vitamina y cada posible aumento de velocidad, actualizando los valores de *dp*. Devuelve la cantidad mínima de vitaminas necesarias para alcanzar G o -1 si no es posible.

La función *main* lee el número de casos de prueba *T*. Para cada caso de prueba, lee el límite de aumento de velocidad (*G*), el número de tipos de vitaminas (*V*), y los aumentos de velocidad de cada vitamina. Llama a la función *minVitamins* y muestra el resultado para cada caso de prueba.