**Pruebas de Software**

PEC1: Prueba de algoritmos que resuelven el Problema de la Mochila

Alumno: Iván Fernández Rodríguez

DNI:34629848J

Correo electrónico: ifernande1631@alumno.uned.es

Centro Asociado: Ourense

Índice

[1. Solución tarea 1: Probar búsqueda\_exhaustiva 3](#_Toc164959818)

[a. Clases de equivalencia y valores límite. 3](#_Toc164959819)

[b. Especificación ACTS de los valores de prueba. 5](#_Toc164959820)

[c. Pruebas de unidades con pytest 5](#_Toc164959821)

[d. Corrección del programa 5](#_Toc164959822)

[2. Solución tarea 2: Prueba búsqueda\_con\_poda y algoritmo\_voraz 5](#_Toc164959823)

[3. Solución tarea 3: Implementación y prueba de una solución basada en programación dinámica 5](#_Toc164959824)

1. **Solución tarea 1: Probar búsqueda\_exhaustiva**
   1. **Clases de equivalencia y valores límite.**

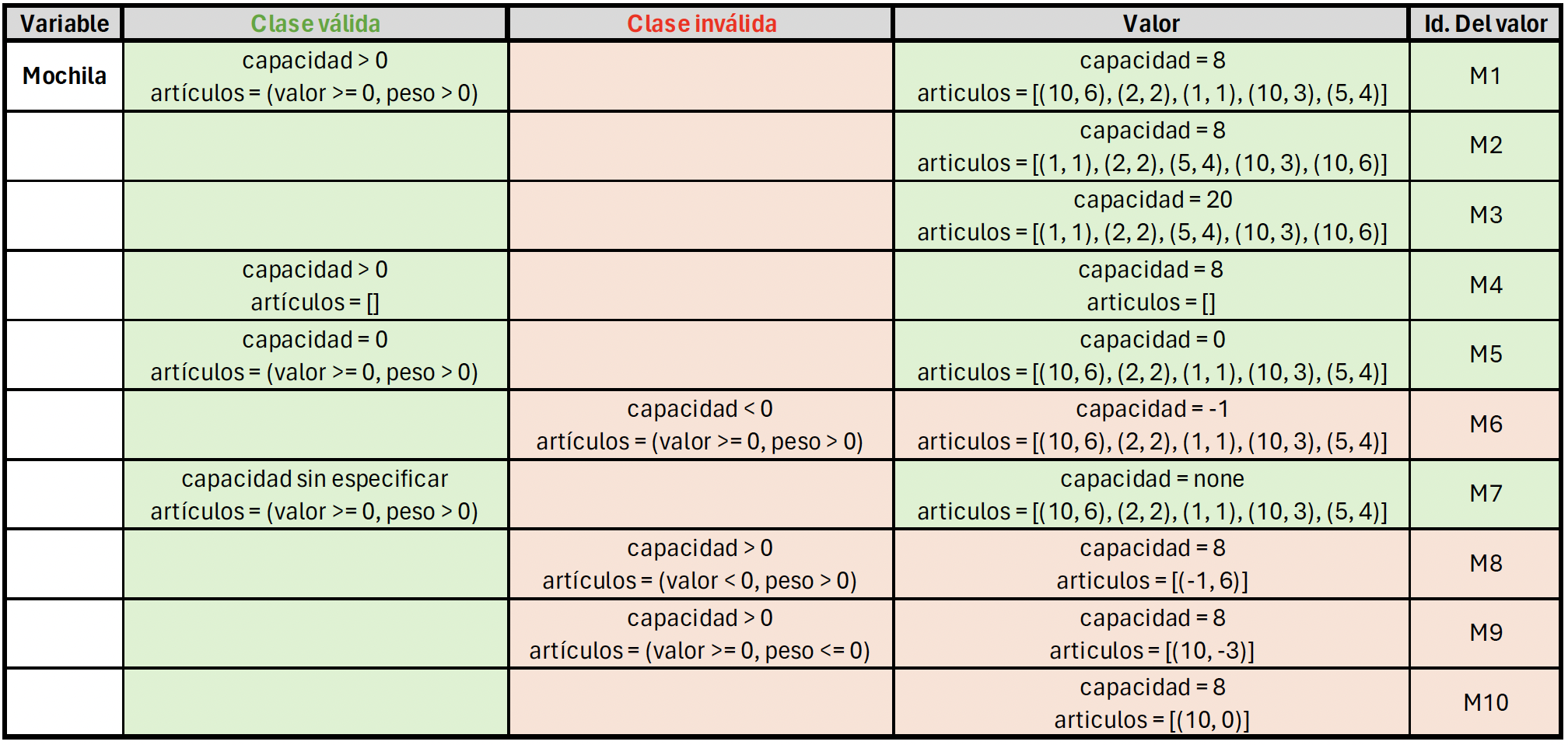
Para determinar las clases de equivalencia y los valores límite he seguido los siguientes pasos:

* **Identificar que variables interesa probar y su tipo**. Solamente tenemos una variable de entrada (mochila), que es una instancia de la clase Mochila.
* **Identificar los atributos de la variable y su tipo**.
  + **Capacidad**: entero
  + **Artículos**: array de instancias de la clase Articulo, cuyos atributos son:
    - **Valor**: entero
    - **Peso**: entero
* **Determinar las dimensiones de cada variable**. Para este paso, consideramos para la variable mochila las siguientes dimensiones:
  + **Capacidad suficiente**. Capacidad para algunos artículos. Capacidad insuficiente para contener algún artículo. Capacidad de sobra para contener todos los artículos.
  + **Longitud array artículos**. Sólo entran algunos artículos en la mochila. Entran todos los artículos en la mochila. Array de artículos vacío.

El objetivo de nuestro SUT es cubrir la capacidad de la variable mochila con los artículos contenidos en el array intentando que la suma de los valores de los artículos seleccionados sea lo mayor posible. Considero, entonces, que los test deben orientarse a estudiar cómo reacciona el SUT frente a distintos valores de capacidad de la mochila, por lo que considero que la dimensión principal de la variable mochila es el valor de la capacidad.

* **Particionar la dimensión principal**. Consideraremos las siguientes particiones:
  + Capacidad > 0
  + Capacidad = 0
  + Capacidad < 0
  + Capacidad sin especificar
* **Determinar si existe alguna relación entre los valores primarios de las variables**. El número de artículos en el array, como el valor o el peso de cada uno de ellos no influye en los valores de la variable capacidad.
* **Diseñar la tabla de clases de equivalencia y valores límite para la dimensión principal**.

Para determinar las clases de equivalencia, hice subconjuntos de los posibles valores de los atributos de la variable de entrada (mochila), y puesto que uno de ellos es un array de pares (valor, peso), donde cada uno corresponde a un artículo, era necesario tener en cuenta, también, estos dos atributos de la clase Artículo, así como la longitud del vector. He dividido el conjunto de valores que puede tomar la variable capacidad en cuatro subconjuntos (capacidad>0, capacidad=0, capacidad<0 y capacidad sin especificar), la variable valor en dos subconjuntos (valor>=0 y valor<0), la variable peso en dos subconjuntos (peso>0, peso<=0) y el número de artículos en dos subconjuntos (longitud>0 y longitud=0).



* **Identificar las restricciones.**

Las restricciones entre las dimensiones de interés podrían incluir:

* + Capacidad de la mochila: debe ser un valor positivo. Debe ser lo suficientemente grande como para permitir la colocación de al menos un artículo, de lo contrario, el algoritmo de búsqueda exhaustiva no tendría sentido.
  + Valor de los artículos: los valores de los artículos deben ser valores no negativos, de lo contrario, invalidaría la necesidad de la búsqueda exhaustiva.
  + Peso de los artículos: deben ser valores positivos y menores o iguales a la capacidad de la mochila, de lo contrario será imposible seleccionarlos.
  + Número de artículos: debe ser un valor entero no negativo. Si es cero, entonces no hay necesidad de ejecutar el algoritmo.
  1. **Especificación ACTS de los valores de prueba.**

He obtenido los dos juegos de pruebas alternativos como indica el enunciado, y los he exportado en formato csv (JP1-output.csv y JP2-output.csv) con el fin de interpretarlos en los test con la librería Pandas de Python.

La principal diferencia entre los dos juegos de prueba es que JP1 contempla valores erróneos en las variables, por lo que se puede utilizar para comprobar que el sistema detecta dichos errores (todos ellos). JP2 no contempla valores erróneos, por lo que es útil para comprobar que se detiene la ejecución del programa en presencia de dichos errores.

* 1. **Pruebas de unidades con pytest**

Aquí.

* 1. **Corrección del programa**

1. **Solución tarea 2: Prueba búsqueda\_con\_poda y algoritmo\_voraz**

As.

1. **Solución tarea 3: Implementación y prueba de una solución basada en programación dinámica**