UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

AÑO ACADÉMICO (2023 – 2024)

SEGUNDO CURSO

EJERCICIO GUIADO 2
DESARROLLO DIRIGIDO POR PRUEBAS

Información Alumno 1

Nombre: Gabriel José

Apellidos: Rivera Amor

NIA: 100477706

Correo: 100477706@alumnos.uc3m.es

Grupo: 84

Titulación: Grado en Ingeniería Informática

Información Alumno 2

Nombre: Joaquín

Apellidos: Pujol Carrillo

NIA: 100495826

Correo: 100495826@alumnos.uc3m.es

Grupo: 84

Titulación: Grado en Ingeniería Informática

Información Alumno 3

Nombre: Daniel

Apellidos: Ortiz Hincapié

NIA: 100499700

Correo: 100499700@alumnos.uc3m.es

Grupo: 84

Titulación: Grado en Ingeniería Informática

Madrid 31 de marzo de 2024

ÍNDICE

DESCRIPCIÓN DE CASOS DE PRUEBA

Caso de la Función 1:

La primera función tiene el objetivo de recibir un conjunto de información relacionada al cliente que está generando la reserva en nuestro hotel y guardarlas en un fichero json. Para lograr esto, es necesario establecer un conjunto de clases de equivalencia y valores límites, los que van a permitir al programa discernir si la información suministrada por el cliente, es correcta.

1. Clases de Equivalencia y Valores Límites para la Variable Tarjeta de Crédito

VARIABLE	CRITERIA	VALID CLASSES	INVALID CLASSSES
CREDIT CARD	Cifra de 16 dígitos que cumpla con el Algoritmo de Luhm	CEV1: 5344 5553 9577 0248	CENV1: Son 16 dígitos, pero uno de ellos no es válido
	Tipo de Dato Introducido	CEV2: 5344 5553 9577 0248	CENV2: No es un número
	Longitud	CEV3: 5344 5553 9577 0248	CENV3: Más de 16 dígitos (17 dígitos o más)
		VLV1: 5344 5553 9577 0248	CENV4: Menos de 16 dígitos (15 dígitos o menos)
			VLNV1: 17 dígitos
			VLNV2: 15 dígitos

Para el caso de la variable correspondiente a la tarjeta de crédito, tenemos clases de equivalencia que se encargarán de verificar su cumple el algoritmo de Luhm, si es un número y si tiene estrictamente 16 dígitos. Si cumple todas estas condiciones de las clases de equivalencia y se queda dentro del rango del valor límite establecido, la tarjeta de crédito, suministrada por el cliente, será válida.

2. Clases de Equivalencia y Valores Límites para la Variable ID Card

VARIABLE	CRITERIA	VALID CLASSES	INVALID CLASSSES
DNI	Cadena de 9 caracteres, con 8 dígitos y un carácter alfabético de control al final	CEV1: 11185346D	CENV1: Son 8 dígitos, pero el último valor no es una letra.
	Tipo de Dato Introducido	CEV2: 11185346D	CENV2: Alguno de los 8 primeros caracteres no son números
			CENV3: El último carácter no es un carácter alfabético
	Longitud	CEV3: 11185346D	CENV4: Más de 8 dígitos (9 dígitos)
		VLV1: 11185346D	CENV5: Menos de 8 dígitos (7 dígitos o menos)
			CENV6: Más de 1 carácter alfabético (2 caracteres o más)
			VLNV1: 10 caracteres
			VLNV2: 8 caracteres

Para el caso de la variable correspondiente al ID Card o DNI, tenemos clases de equivalencia que se encargarán de verificar que el contenido del DNI sean primero 8 dígitos y el último carácter, es decir, el noveno, sea una letra del alfabeto sin contar la “ñ”. También verificará que haya estrictamente 8 dígitos y estrictamente una letra al final de la cadena. Por último, se tendrán los valores límites que verificarán que la cadena tenga sólo 9 caracteres. Si cumple todas estas condiciones de las clases de equivalencia y se queda dentro del rango del valor límite establecido, el DNI, suministrado por el cliente, será válida.

3. Clases de Equivalencia y Valores Límites para la Variable Nombre y Apellido

VARIABLE	CRITERIA	VALID CLASSES	INVALID CLASSSES
NAME AND SURNAME	Cadena de entre 10 y 50 caracteres con al menos 2 cadenas separadas por un espacio en blanco.	CEV1: Gabriel Rivera	CENV1: Tiene entre 10 a 50 caracteres, pero uno de ellos no es de carácter alfabético
			CENV2: Tiene entre 10 y 50 caracteres, pero es una misma cadena de caracteres, sin espacios
	Tipo de Dato Introducido	CEV2: Gabriel Rivera	CENV3: No es una cadena de caracteres
			CENV4: No hay ningún carácter " ", que represente el espacio entre dos cadenas de caracteres
	Longitud	CEV3: Gabriel Rivera	CENV5: Más de 50 caracteres (51 caracteres o más)
		VLV1: Gabriel Rivera	CENV6: Menos de 10 caracteres (9 caracteres o menos)
			CENV7: Menos de 1 carácter en blanco en la cadena
			VLNV1: 51 caracteres
			VLNV2: 9 caracteres

Para el caso de la variable correspondiente al Nombre y Apellido, tenemos clases de equivalencia que se encargarán de verificar que esta cadena de caracteres tenga entre 10 y 50 caracteres de extensión. También verificará que la cadena este formada únicamente por caracteres y que tengan al menos un espacio en blanco. Por último, se toman en cuenta los valores límites, sabiendo que una cadena de 51 caracteres o 9, es inválida, y que una cadena con 11 caracteres o 49 es válida. Si cumple todas estas condiciones de las clases de equivalencia y se queda dentro del rango del valor límite establecido, el Nombre y Apellido, será válido.

4. Clases de Equivalencia y Valores Límites para la Variable Número de Teléfono

VARIABLE	CRITERIA	VALID CLASSES	INVALID CLASSSES
PHONE NUMBER	Cifra de 9 dígitos, todos caracteres numéricos	CEV1: 613589749	CENV1: Son 9 términos, pero uno de ellos no es un numéro
	Longitud	CEV2: 613589749	CENV2: Más de 9 dígitos (10 dígitos o más)
		VLV1: 613589749	CENV3: Menos de 9 dígitos (8 dígitos o menos)
			VLNV1: 10 dígitos
			VLNV2: 8 dígitos

Para el caso de la variable correspondiente al número de teléfono, tenemos clases de equivalencia que se encargarán de verificar que sean solo 9 dígitos, ni más, ni menos, y que ninguno de ellos sea distinto a un dígito. Si cumple todas estas condiciones de las clases de equivalencia y se queda dentro del rango del valor límite establecido, el número de teléfono, suministrado por el cliente, serán válidos.

5. Clases de Equivalencia y Valores Límites para la Variable Tipo de Habitación

VARIABLE	CRITERIA	VALID CLASSES	INVALID CLASSSES
ROOM TYPE	Símbolo Identificativo que puede ser: *Single* (O), *Double* (D) o *Suit* (S)	CEV1: *Single*	CENV1: El dato introducido es distinto a Single, Double o Suit
		CEV2: *Double*	
		CEV3: *Suit*	
	Tipo de Dato Introducido	CEV4: *Single*	CENV2: El dato introducido no es una cadena de caracteres

Para el caso de la variable correspondiente al tipo de habitación, tenemos clases de equivalencia que se encargarán de verificar que lo que se ingresa por parte del cliente, sea “Single”, “Double” o “Suit”. Si no es alguna de estas, sería un caso inválido. Si cumple todas estas condiciones de las clases, tipo de habitación, suministrado por el cliente, será válido.

6. Clases de Equivalencia y Valores Límites para la Variable Número de Días

VARIABLE	CRITERIA	VALID CLASSES	INVALID CLASSSES
NUM DAYS	Cifra con un valor entre 1 y 10	CEV1: 2	CENV1: El dígito introducido no es válido
	Tipo de Dato Introducido	CEV2: 2	CENV2: No es un número
	Rango de Valores	CEV3: 2	CENV3: El dígito es mayor a 10
		VLV1: 2	CENV4: El dígito es menor a 1
			VLNV1: 11
			VLNV2: 0

Para el caso de la variable correspondiente al Número de Días, tenemos clases de equivalencia que se encargarán de verificar que lo que se introduzca sea un número y que se encuentre dentro del rango entre 1 y 10. Esto también se verifica en los valores límites, donde aquellos que sean menor a 1 y mayores a 10, serán valores inválidos. Si cumple todas estas condiciones de las clases de equivalencia y se queda dentro del rango del valor límite establecido, del Número de Días, será válido.

7. Clases de Equivalencia y Valores Límites para la Variable Fecha de Llegada

VARIABLE	CRITERIA	VALID CLASSES	INVALID CLASSSES
ARRIVAL DATE	Cadena de caracteres en el formato "DD/MM/YYYY", la cual tiene 8 dígitos, donde después de los dos primeros, hay un separador. Lo mismo ocurre después de los 4 dígitos	CEV1: 27/02/2025	CENV1: La cadena de caracteres es correcta, pero tiene un carácter no válido
			CENV2: Tenemos 8 dígitos sin separadores
	Valores para DD, MM y YYYY	CEV2: 01/01/2025	CENV3: DD tenga un número con valor menor a 01 (0 o menor)
			CENV4: DD tenga un número con valor mayor a 30 (31 o mayor)
			CENV5: MM tenga un número con valor menor a 01 (0 o menor)
			CENV6: MM tenga un número con valor mayor a 12 (13 o mayor)
			CENV7: La fecha establecida es menor a la fecha actual del sistema
	Formato	CEV3: 27/02/2025	CENV8: 3 o más dígitos antes de la primera "/ "
			CENV9: Después de la primera "/ " y antes de la segunda, hayan 3 o más dígitos
			CENV10: Después de la segunda "/ " hayan 5 o más dígitos
	Longitud	VLV1: 27/02/2025	VLNV1: 11 caracteres
			VLNV2: 9 caracteres

Para el caso de la variable correspondiente a la Fecha de Llegada, tenemos clases de equivalencia que se encargarán de verificar que existan 8 dígitos, donde después de los dos primeros, haya un separador, luego se tengan otros dos dígitos, luego otro separador y por último cuatro dígitos al final. También verifica que los primeros dos dígitos, la combinación de ellos, no debe ser menor a 30 y mayor a 01. Luego, la combinación de los meses, no debe ser menor a 01 y no mayor a 12. Por último, estas clases de equivalencia garantizan la fecha de llegada, sea mayor que la fecha del sistema. Si cumple todas estas condiciones de las clases de equivalencia y se queda dentro del rango del valor límite establecido, del Día de Llegada, será válido.

8. Clases de Equivalencia y Valores Límites para la Variable Localizer

VARIABLE	CRITERIA	VALID CLASSES	INVALID CLASSSES
LOCALIZER	Cadena de caracteres en hexadecimal con 32 caracteres	CEV1: abcdef123456789ab123456789cdfa	CENV1: Exception CREDIT_CARD
			CENV2: Exception DNI
			CENV3: Exception NAME_SURNAME
			CENV4: Exception PHONE_NUMBER
			CENV5: ROOM_TYPE
			CENV6: NUM_DAYS
			CENV7: ARRIVAL_DATE
			CENV8: Un mismo cliente que tenga una reserva para la misma fecha

Para el caso de la variable correspondiente a la Localizer, tenemos clases de equivalencia que se encargarán de verificar que si no se produce ningún error en las variables introducidos por el cliente. Además, mediante una de las clases de equivalencia, debe verificar que, bajo las informaciones suministrada por el cliente, éste no tenga ya una reserva realizada. Si todo está en orden, se realiza entonces la reserva y genera un Localizer automáticamente.

Caso de la Función 2:

La segunda función tiene el objetivo de generar un número identificativo de la llave de acceso a la habitación, así como un archivo que especifica la fecha de llegada y la de salida. Para ello, la función recibe un archivo json con el ID Card y un Localizer. La función debe verificar que el Localizer suministrado, esté en el documento de reserva y además ese Localizer, coincida con el ID Card que realizó dicha reserva. Además, debe comprobar que la fecha de llegada que se genera en esta función, contrasta con la fecha establecida en la reserva.

Para realizar todas estas comprobaciones, es necesario analizar el fichero de entrada json y verificar que el fichero tenga la estructura adecuada, por lo que es necesario realizar el análisis sintáctico de dicho fichero, teniendo en consideración lo siguiente.

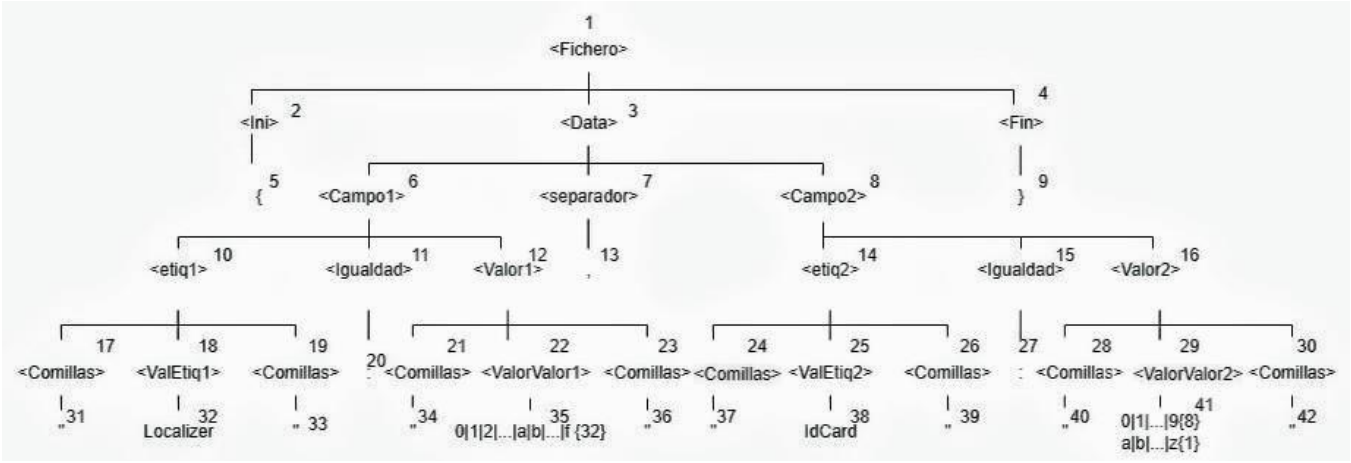
1. Gramática de la Segunda Función

{ “Localizer”:”<string de 32 en hexadecimal>”, “IdCard”:”<DNI válido>” }

- Fichero ::= Ini Data Fin
- Ini ::= {
- Fin ::= }
- Data ::= Campo1 Separador Campo2
- Campo1 ::= Etiq1 Igualdad Valor1
- Campo2 ::= Etiq2 Igualdad Valor2
- Igualdad ::= :
- Separador ::= ,
- Etiq1 ::= Comillas Valetiq1 Comillas
- Valor1 ::= Comillas Valorvalor1 Comillas
- Valetiq1 ::= Localizer
- Valorvalor1 ::= 0|1|2|...|9|a|...|f {32}

- Comillas ::= “
- Etiq2 ::= Comillas Valetiq2 Comillas
- Valor1 ::= Comillas Valorvalor2 Comillas
- Valetiq2 ::= Localizer
- Valorvalor2 ::= 0|1|2|...|9 {8} |a|...|z {1}

2. Árbol de Derivación



Para verificar que se cumpla la estructura del fichero de entrada es necesario realizar la duplicación y el borrado de los nodos no terminales del árbol de derivación, generado a partir de la gramática principal. El árbol se debe leer de arriba abajo y de izquierda a derecha. Estas pruebas implican repetir el contenido del nodo no terminal o en el otro caso, la eliminación del contenido del nodo que estamos analizando.

Este proceso debemos hacerlo con los nodos 1, 2, 3, 4, 6, 7, 8, 10, 11, 12, 14, 16, 17, 18, 22, 25, 29. Es importante mencionar que, el nodo 11 es igual al nodo 15, por lo que las pruebas de borrado y duplicado para el nodo 15, son las mismas que las del nodo 11, por lo que no es necesario repetirlas. Lo mismo ocurre con el nodo 17 y los nodos 19, 21, 23, 24, 26, 28, 30.

También debemos hacer el borrado, duplicado y modificado de los nodos terminales, los cuales son los nodos 5, 9, 13, 20, 31, 32, 35, 38, 41. Es importante mencionar que, el nodo 20 es igual al nodo 27, por lo que las pruebas de modificado para el nodo 27, son las mismas que las del nodo 20, por lo que no es necesario repetirlas. Lo mismo ocurre con el nodo 31 y los nodos 33, 34, 36, 37, 39, 40, 42. Es importante mencionar que la modificación de un nodo terminal, es cambiar el símbolo terminal, lo cual debería arrojar un mensaje de error al modificar.

Además, en el caso de los nodos terminales, solo es necesario realizar las pruebas de modificación, dados que las de borrado y duplicado, se llevan a cabo directamente con el borrado y duplicación de los nodos no terminales que los generan.

Si se aplica alguna de las condiciones anteriores al fichero introducido en la función, se arroja un error al momento de la decodificación del archivo. Estas pruebas se encuentran mayormente especificada en el documento Excel anexo a este documento.

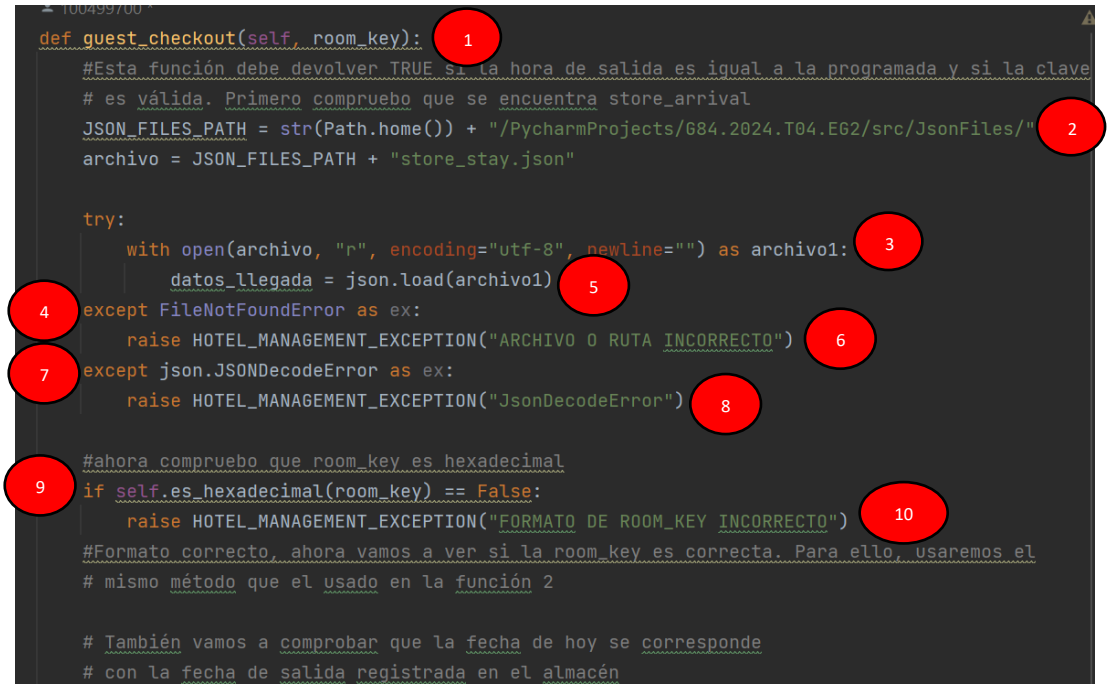
Una vez se ha determinado la validez del archivo introducido con las pruebas anteriores de sintaxis, entonces se proceden a realizar los otros casos de prueba, con tal de poder generar la llave de la habitación y generar un archivo que especifica el día de la salida del cliente.

Caso de la Función 3:

La tercera función tiene el objetivo de generar un archivo que guarde los datos de salida de un cliente del hotel. Para ello, a esta función se le ingresará como parámetro, el código de una llave de una habitación y la función deberá contrastar que la fecha actual y la fecha de salida establecida en el fichero generado en la función 2, coinciden.

Dentro del archivo que almacena los códigos de habitaciones, así como aquel que almacena las reservas, puede haber infinidad de registros, por lo que es necesario realizar un diagrama de flujo de esta función, con tal de determinar los caminos posibles que se pueden dar en ella. Una vez generados estos caminos, es necesario recorrer cada uno de ellos, para verificar que la función funcione en cualquiera de los casos.

1. **Nodos en el Código**

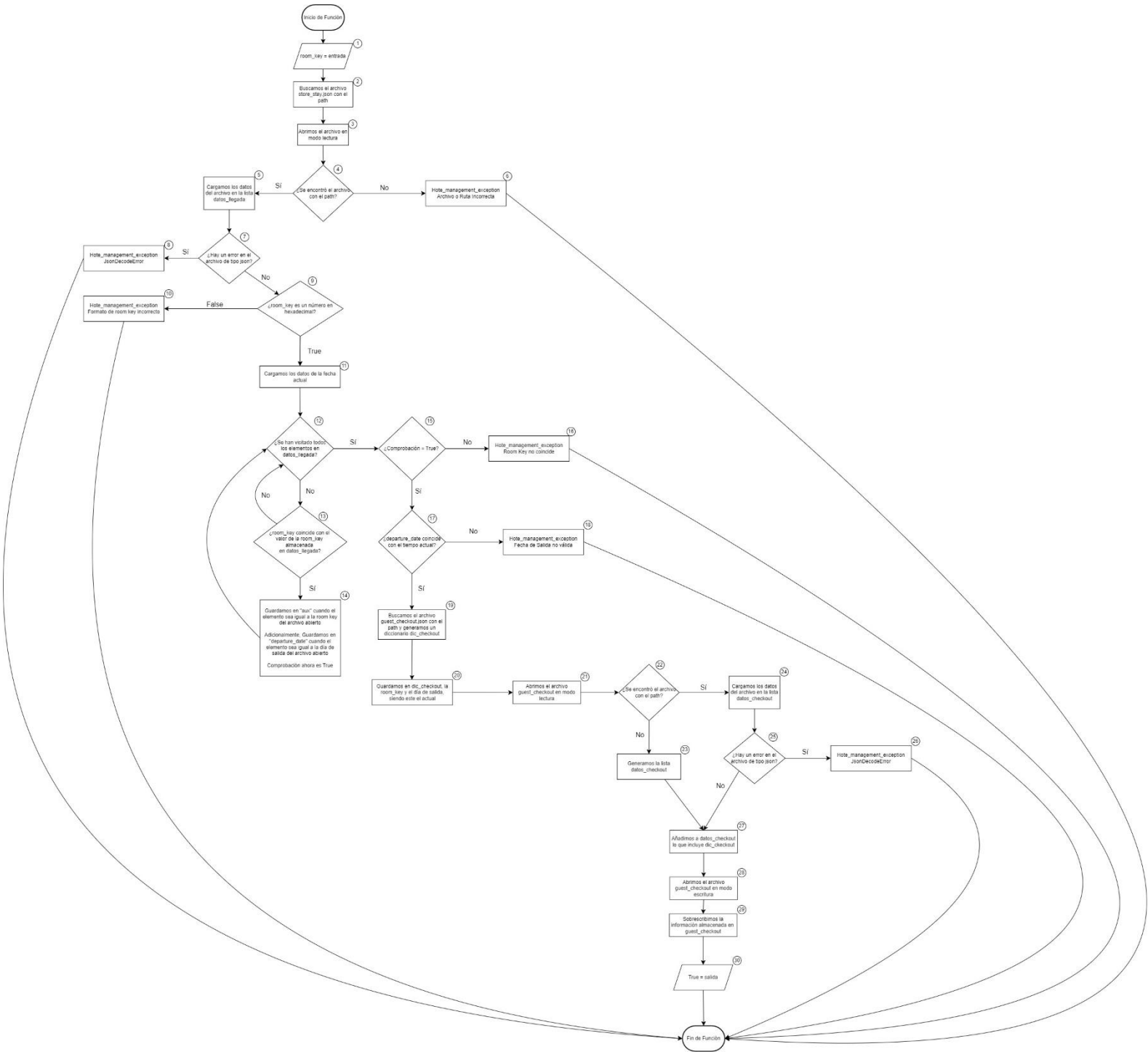



```
def guest_checkout(self, room_key):
    #Esta función debe devolver TRUE si la hora de salida es igual a la programada y si la clave
    # es válida. Primero compruebo que se encuentra store_arrival
    JSON_FILES_PATH = str(Path.home()) + "/PycharmProjects/G84.2024.T04.EG2/src/JsonFiles/"
    archivo = JSON_FILES_PATH + "store_stay.json"

    try:
        with open(archivo, "r", encoding="utf-8", newline="") as archivo1:
            datos_llegada = json.load(archivo1)
    except FileNotFoundError as ex:
        raise HOTEL_MANAGEMENT_EXCEPTION("ARCHIVO O RUTA INCORRECTO")
    except json.JSONDecodeError as ex:
        raise HOTEL_MANAGEMENT_EXCEPTION("JsonDecodeError")

    #ahora compruebo que room_key es hexadecimal
    if self.es_hexadecimal(room_key) == False:
        raise HOTEL_MANAGEMENT_EXCEPTION("FORMATO DE ROOM_KEY INCORRECTO")
    #Formato correcto, ahora vamos a ver si la room_key es correcta. Para ello, usaremos el
    # mismo método que el usado en la función 2

    # También vamos a comprobar que la fecha de hoy se corresponde
    # con la fecha de salida registrada en el almacén
```

```
    justnow = datetime.utcnow()
    current_datetime = justnow.strftime('%d/%m/%Y %H:%M')
    departure_date = str
    llave = str
    comprobacion = False

    try:
        for element in datos_llegada:
            if room_key == element["_HOTEL_STAY__room_key"]:
                llave = element["_HOTEL_STAY__room_key"]
                departure_date = element["_HOTEL_STAY__departure"]
                comprobacion = True
    if comprobacion != True:
            raise HOTEL_MANAGEMENT_EXCEPTION("ROOM KEY NO COINCIDE")
    if current_datetime[:10] != departure_date[:10]:
            raise HOTEL_MANAGEMENT_EXCEPTION("FECHA DE SALIDA NO VÁLIDA")
```

```
JSON_FILES_PATH = str(Path.home()) + "/PycharmProjects/G84.2024.T04.EG2/src/JsonFiles/"
checkout = JSON_FILES_PATH + "guest_checkout.json"

dic_checkout = {}
dic_checkout.update({'_CHECKOUT__room_key': llave,
                     '_CHECKOUT__departure_time': current_datetime})
try:
    with open(checkout, "r", encoding="utf-8", newline="") as archivo1:
        datos_checkout = json.load(archivo1)
except FileNotFoundError as ex:
    datos_checkout = []
except json.JSONDecodeError as ex:
    raise HOTEL_MANAGEMENT_EXCEPTION("JsonDecodeError")

datos_checkout.append(dic_checkout)

with open(checkout, "w", encoding="utf-8", newline="") as file:
    json.dump(datos_checkout, file, indent=2)
print("Checkout Efectuado")

return True
```

2. Diagrama de Flujo



3. Casos de Prueba

Para verificar la funcionalidad de nuestra última función, fue necesario establecer los caminos que se recorrerían, con la finalidad de cubrir todos los posibles casos que pudieran surgir. Como se puede evidenciar en el Excel adjunto a este documento, se determinaron 14 posibles caminos, de cuales, cinco de ellos se corresponden a la verificación del bucle.

El primer camino es aquel que permite recorrer todo el diagrama de flujo, pasando por la mayor cantidad de nodos del mismo, pero considerando que el fichero json, que almacena la información de los checkouts, no existe y es necesario crearlo. Este camino, incluye los nodos 1-2-3-4-5-7-9-11-12-13-14-12-15-17-19-20-21-22-23-27-28-29-30-F, del diagrama.

El segundo camino es aquel que permite recorrer todo el diagrama de flujo, pasando por la mayor cantidad de nodos del mismo, pero considerando que el fichero json, que almacena la información de los checkouts, tiene contenido dentro de él, por lo que es necesario cargarlo y verificar que no tiene un error de decodificación. Este camino, incluye los 1-2-3-4-5-7-9-11-12-13-14-12-15-17-19-20-21-22-24-25-27-28-29-30-F, del diagrama.

El tercer camino es aquel que permite recorrer todo el diagrama de flujo, pasando por la mayor cantidad de nodos del mismo, pero considerando un error. Como se mencionó anteriormente, en el caso de que el fichero json, que almacena la información de los checkouts, tiene contenido dentro de él, es necesario cargar dicho contenido y verificar que no tiene un error de decodificación. Esta prueba, se encarga de cubrir el camino o la desviación que se da cuando hay un error al cargar de formato del archivo. Este camino, incluye los 1 1-2-3-4-5-7-9-11-12-13-14-12-15-17-19-20-21-22-24-25-26-F, del diagrama.

Es importante acotar que, en este caso, así como en el caso correspondiente al camino 15, en las pruebas unitarias en Python, no se pudieron probar estos casos, ya que la ruta de dónde buscar el archivo se encontraba dentro del código. Este imposibilitaba entonces, de hacer lo mismo que se realizó en las pruebas de la segunda función y redirigir la prueba a un fichero json con errores.

Siguiendo con la especificación de caminos, el cuarto camino es aquel que permite recorrer todo el diagrama de flujo, pero con un error en la fecha de salida. Este es un caso de desviación, necesario de cubrir, derivado del nodo 17, y donde se verifica si la fecha actual coincide con la fecha de salida estipulada en el archivo de la función 2. Este camino, incluye los 1 1-2-3-4-5-7-9-11-12-13-14-12-15-17-18-F, del diagrama.

Para el quinto camino, se tiene un camino que permite recorrer todo el diagrama de flujo, pero con un error en el código de la llave de la habitación. Este es un caso de desviación, necesario de cubrir, derivado del nodo 15, que a su vez está íntimamente relacionado al condicional dentro del bucle (nodo 13). Este camino, permite determinar si la llave que se ha introducido como

parámetro, contrasta con el código que se encuentra dentro del archivo de la función 2. Este camino, incluye los 1-2-3-4-5-7-9-11-12-13-14-12-15-16-F, del diagrama.

El sexto camino, es el primer caso que se deriva de la realización del único bucle de la función y tiene como objetivo que el programa evite el bucle y su ejecución. Si esto ocurre, entonces el resultado sería el mismo que el del anterior camino, por lo que daría un error en el código de la llave, dado que la variable "comprobación" del nodo 15, tendría un valor booleano de "False". Por ende, el sexto camino sería 1-2-3-4-5-7-9-11-12-15-16-F

Luego, el séptimo camino es aquel que verifica cuando el bucle tiene una sola iteración. Es igual al primer caso de prueba llevado a cabo y la única forma de garantizar que el bucle itere una vez, es que el archivo de la función 2 tenga un solo código de llave y la información de una sola reserva. Por ende, su camino sería 1-2-3-4-5-7-9-11-12-13-14-12-15-17-19-20-21-22-23-27-28-29-30-F.

El octavo camino es aquel que verifica cuando el bucle tiene dos iteraciones. La única forma de garantizar que el bucle itere dos veces, es que el archivo de la función 2 tenga dos códigos de llave y la información de dos reservas, siendo segunda la que tiene la información deseada. De esta forma, como el bucle se recorrerá mientras haya elementos en el fichero de la función dos, cuando llegue a la segunda reserva, encontrará el código buscado y saldrá del bucle. Por ende, su camino sería 1-2-3-4-5-7-9-11-12-13-12-13-14-12-15-17-19-20-21-22-23-27-28-29-30-F.

Por su parte el noveno camino es aquel que verifica cuando el bucle tiene el máximo de iteraciones menos una. La única forma de garantizar que el bucle itere este número de veces, es que el archivo de la función 2 tenga un número definido de códigos de llave y la información buscada esté en el último de los registros. De esta forma, el bucle se recorrerá mientras haya elementos en el fichero de la función 2 y su camino sería 1-2-3-4-5-7-9-11-12-13-12-13-12-13-12-13-14-12-15-17-19-20-21-22-23-27-28-29-30-F.

Es importante mencionar que como el número máximo de iteraciones no está definido, ya que depende del número de reservas que contenga el fichero de reservas, entonces nosotros, a discreción, colocamos como número máximo de registros, el valor de cinco. Esto, con la finalidad de llevar a cabo las pruebas y teniendo en consideración que cinco pruebas es muestra representativa de la capacidad máxima del fichero, sin dejar de considerar que puede haber muchas más.

Siguiendo con las pruebas, el décimo camino es aquel que verifica cuando el bucle tiene el máximo de iteraciones. La única forma de garantizar que el bucle itere este número de veces, es que el archivo de la función 2 tenga un número definido máximo de códigos de llave y la información buscada esté en el último de los registros. De esta forma, el bucle se recorrerá

mientras haya elementos en el fichero de la función 2 y su camino sería 1-2-3-4-5-7-9-11-12-13-12-13-12-13-12-13-12-13-14-12-15-17-19-20-21-22-23-27-28-29-30-F.

El décimo primer camino es aquel que verifica cuando el bucle tiene el máximo de iteraciones más una. La única forma de garantizar que el bucle itere este número de veces, es que el archivo de la función 2 tenga un número definido máximo, más un registro extra, de códigos de llave y la información buscada esté en el último de los registros. De esta forma, el bucle se recorrerá mientras haya elementos en el fichero de la función 2 y su camino sería 1-2-3-4-5-7-9-11-12-13-12-13-12-13-12-13-12-13-12-13-12-13-14-12-15-17-19-20-21-22-23-27-28-29-30-F.

Por otro lado, el décimo segundo camino, es aquel que verifica que el código de llave, que es introducido como parámetro, es hexadecimal y si no lo es, se deriva en esta vertiente. Este camino se aplica entonces cuando el valor que recibe la función como parámetro, no tiene el formato correcto. Su camino está formado por los nodos 1-2-3-4-5-7-9-10-F.

El décimo tercer camino, es aquel que se da cuando al cargar los datos generados por la función 2, que le precede a esta, hay un error de codificación. El camino correspondiente a este caso es 1-2-3-4-5-7-8-F

Por último, pero no menos importante, el décimo cuarto camino, es aquel que se deriva cuando no se puede encontrar el archivo que se produce como resultado de la segunda función. Esta situación, se puede provocar porque ha ocurrido un error en la función anterior o no se ha hecho ninguna reserva al ejecutar esta función. Por ende, la función avisa que no se pudo encontrar el archivo solicitado y se tiene el siguiente camino 1-2-3-4-6-F

CONCLUSIONES

Para finalizar con este documento, y con él, la especificación de todos los casos de prueba, podemos culminar diciendo que hemos aprendido inmensamente de este trabajo, teniendo que considerar la gestión de las tareas, el tiempo de ejecución y la complejidad de los apartados.

Es importante decir que la programación en equipo nos ayudó a tener una perspectiva más amplia de todo lo que implica el proceso de programar y probar un determinado programa, utilizando herramientas del desarrollo de software, con la finalidad de que este sea robusto y cumpla su función. Por ello, es de decir que más allá del desarrollo de las funciones, podemos decir que este trabajo amerita un gran tiempo de dedicación para que todas las partes encajen correctamente y todo funcione como debería, siendo esto, lo más complicado del mismo.

Fuera de esos comentarios, nos sentimos muy orgullosos del trabajo realizado y del esfuerzo puesto sobre este ejercicio, dando por hecho que el esfuerzo valió la pena.