Diseño de Pruebas: Fecha

1. Identificar las unidades de prueba

La unidad de prueba principal es la clase Fecha.

2. Identificar las variables de interés

Identificamos las siguientes variables de entrada que influyen en la lógica de negocio y la toma de decisiones del sistema:

1. **día (int):** determina el día del mes.
2. **mes (int):** determina el mes del año.
3. **anio (int):** determina el año.

3. Identificar los valores de prueba para cada una de las variables anteriores

Parámetros	Clases de equivalencia	Valores seleccionados
dia: int	(-∞, 0], [1, 31], [32, +∞)	-1, 0, 2, 17, 31, 32, 45, "a", 25x10^20
mes: int	(-∞, 0], [1, 12], [13, +∞]	-1, 0, 1, 12, 13, 15, "b", 25x10^2014
anio: int	(-∞, -1], [0,], (1, +∞)	-10, -1, 0, 1, 203, "mil", 34x10^34

4. Calcular el número máximo posible de casos de pruebas que se podrían generar a partir de los valores de prueba (combinatoria)

Para calcular el máximo posible de casos de prueba debemos calcular el producto cartesiano (todas las combinaciones posibles).

Total = Ndia x Nmes x Nanio

Total = 9x8x7= 504 combinaciones posibles

5. Cobertura Each Use (Cada Uso): Definir un conjunto de casos de prueba para cumplir con each use

Para implementar cobertura each use, vamos a entender primero lo que significa. Consiste en asegurarse de probar cada valor alguna vez, de esta manera garantiza que, si un valor concreto hace fallar el programa, lo detectaremos.

El número de casos de prueba lo determina la variable que tenga más valores, en nuestro caso, dia.

(Tabla adjunta creada en Excel, para visualizar mejor todos los casos)

ID Caso	dia	mes	anio
CP01	-1	-1	-10
CP02	0	0	-1
CP03	2	1	0
CP04	17	12	1
CP05	31	13	203
CP06	32	15	"diez mil"
CP07	45	"b"	34x10^34
CP08	"a"	25x10^2014	-10 (rep)
CP09	25x10^20	-1 (rep)	-1 (rep)

6. Cobertura pairwaise: Definir un conjunto de pruebas para cumplir pairwaise

Debido a la gran combinatoria que produce pairwaise, se ha utilizado la herramienta PICT sugerida en el enunciado para generar la cobertura pairwaise:
https://pairwise.teremokgames.com/

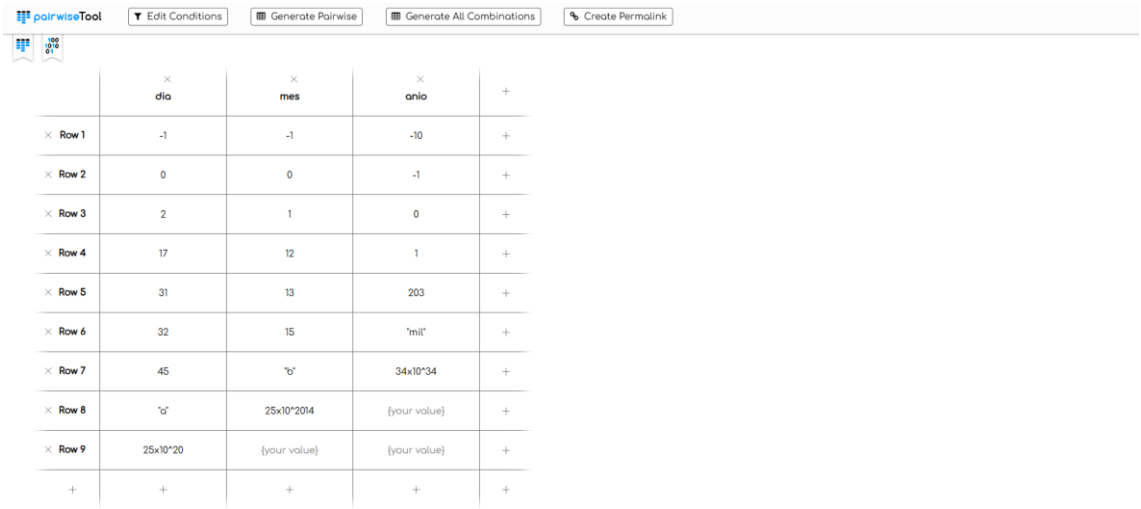
Pairwaise exige que cada pareja posible de valores entre dos variables cualesquiera aparezca al menos una vez.

dia: 9 valores

mes: 8 valores

Solo para cruzar estas 2 variables necesitaríamos un mínimo de 72 casos de prueba. Es decir, la tabla resultante tendrá al menos 72 filas, como realizar esto de manera manual es muy tedioso, tal y como indica el enunciado, se permite hacer uso de una herramienta que calcule la cobertura pairwaise.

A la herramienta se le ha introducido todos los valores definidos en el apartado 3:



Proporcionando el fichero .csv correspondiente:

Pairwiseproblema1.xlsx

Donde se han generado 76 filas, lo que se ajusta a nuestro planteamiento, un mínimo obligatorio de 72 filas.

7. Cobertura de decisiones: definir conjunto de casos de prueba para trozos de código que incluyan decisiones

Primera decisión

A: mes <1 B: mes >12

A	B	A or B	mes
V	F	V	0
F	V	V	13
F	F	F	6

Puesto que es matemáticamente imposible que A=V y B=V, se omite esa fila para no crear confusión

Segunda decisión

A: día <1 B: día >31

A	B	A or B	dia
V	F	V	0
F	V	F	32
F	F	F	19

De igual manera que para la decisión 1, evitamos la posibilidad de A=V y B=V para evitar confusión

(Aunque evaluemos las decisiones por separado, como la primera decisión generaría una excepción en caso V, asumiríamos que tomamos la decisión F, que nos permite avanzar hasta la siguiente decisión, que también puede lanzar excepción)

Tercera decisión

A: anio

A	A	anio
V	V	-1
F	F	19

8. Cobertura MC/DC: definir un conjunto de pruebas para los trozos de código que incluyes decisiones

Primera decisión

A: mes <1 B: mes >12

A	B	A or B	Condición dominante	mes
V	F	V	A	0
F	V	V	B	13
F	F	F	A, B	6

Segunda decisión

A: día <1 B: día >31

A	B	A or B	Condición dominante	dia
V	F	V	A	0
F	V	F	B	32
F	F	F	A, B	19

Tercera decisión

A: anio

A	A	Condición dominante	anio
V	V	A	-1
F	F	A	19

9. Resultados del número de los casos de prueba para los apartados 3, 4 y 5

Con los valores seleccionados en el apartado 3, pudimos generar los casos de prueba para determina la cobertura alcanzada.

En el apartado 4 (combinatoria), obtuvimos un total de 15.598.144 posibles. Lo que imposibilita realizar pruebas exhaustivas en un tiempo razonable.

Con la cobertura each use (apartado 5), conseguimos reducir el conjunto de casos de prueba a 12 (variable más restrictiva, más valores). A pesar de ser muy eficiente en cuanto a tiempo, es una cobertura débil en cuanto a calidad, solo garantiza que se prueban valores individuales, pero no como interactúan entre si, por lo que, en el apartado 6, cobertura pair waise, a pesar de obtener 139 casos de prueba, nos garantizaba que cualquier interacción entre dos variables, ha sido probada al menos una vez.

Respecto a la cobertura alcanzada, podemos afirmar que hemos alcanzado una cobertura robusta funcional, ya que, tenemos garantía de que si existe un fallo provocado por la combinación de dos valores cualesquiera, nuestro conjunto de pruebas lo detectará.

Sin embargo, para asegurar calidad total, realizamos cobertura de decisiones y MC/DC, las cuales garantizan, que no solo se prueban las combinaciones de datos, si no que se recorren todas las ramas del código, validando la independencia de las condiciones lógicas.

Respecto a la medida en la que la implementación del programa influye en el diseño e implementación de los casos de prueba, al usar if con return (no usamos else if) facilitamos el diseño de pruebas de decisiones y MC/DC porque las rutas no son independientes, es decir, si la condición se cumple, termina.

Esto también obliga a diseñar casos de prueba en un orden específico de valores, o con valores concretos para lograr “profundidad”, ya que podría darse el caso de nunca cubrir la lógica final. Lo que demuestra que conocer la estructura interna es vital para alcanzar una buena cobertura de decisiones.