Pruebas de Caja Blanca

"Sistema de pagos de alícuotas"

Integrantes:
Lucas Góngora
Gabriel Manosalvas
Jairo Molina
Yandry Vélez

Fecha 2025-01-23

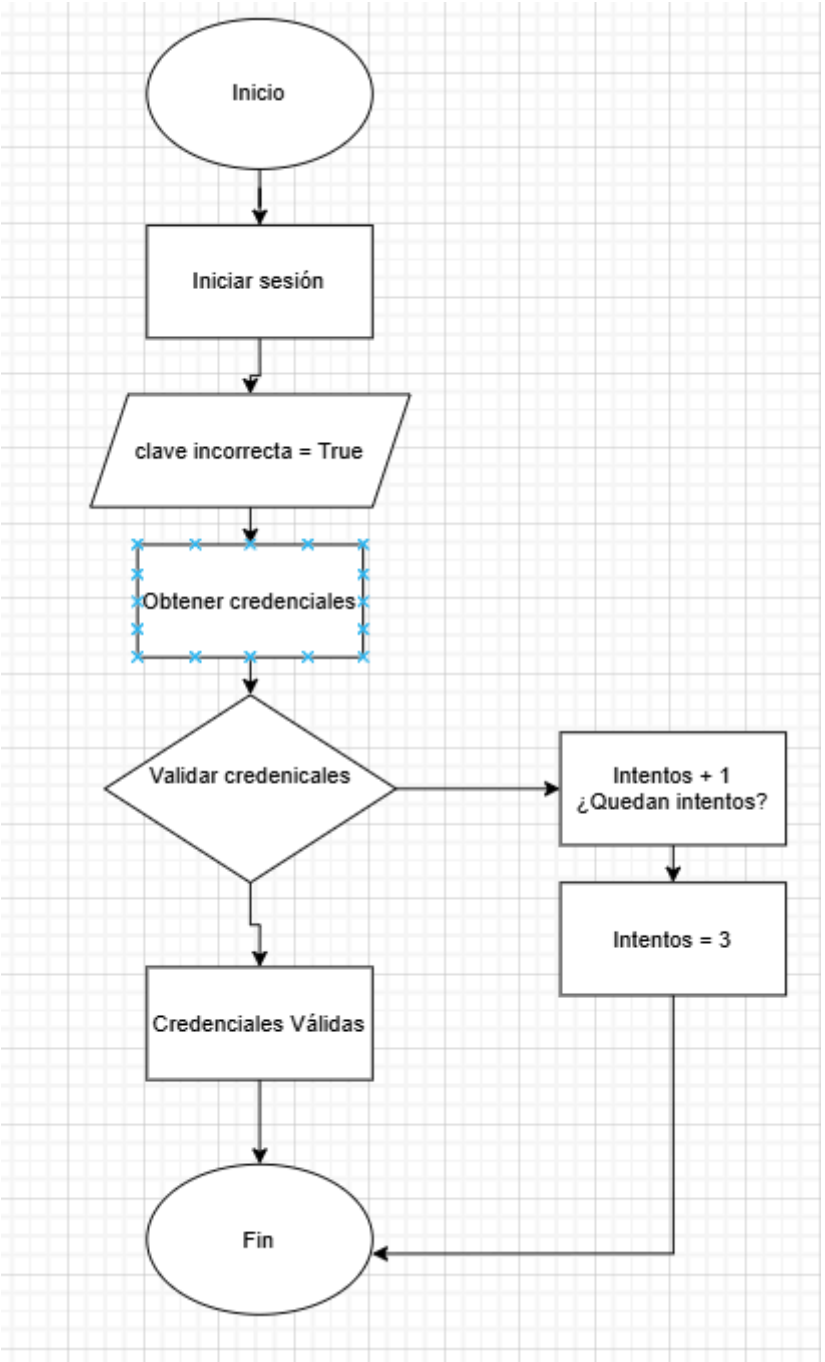
Prueba caja blanca de RF 1.Login

1. **Código FUENTE**

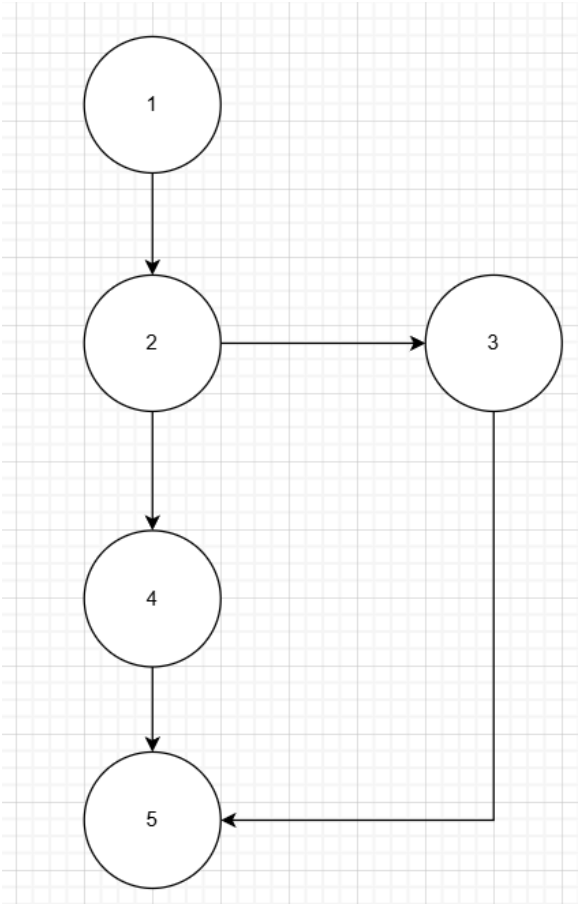
```
import csv
import sys



def iniciar_sesion():
    clave_incorrecta = True
    intentos_realizados = 0
    intentos_permitidos = 3

    while clave_incorrecta:

        usuario,clave = obtener_credenciales()

                                        validacion_exitosa     =
validar_credenciales(usuario,clave)

        if validacion_exitosa:

             clave_incorrecta = False

             print("\nCredenciales validas")

        else:
             print("\nCredenciales invalidas\n")

             intentos_realizados += 1

               print ("Le quedan "+ str(intentos_permitidos -
intentos_realizados) + " intentos")

             if intentos_realizados >= intentos_permitidos:

                        print("Numero de intentos permitidos
alcanzados")

                  sys.exit()



def obtener_credenciales():
    usuario=input("Ingresa tu usuario: ")
```

```
    clave=input("Ingresa tu clave: ")
    return [usuario,clave]


def validar_credenciales(usuario,clave):
                    usuario_autorizado,clave_autorizada     =
credenciales_autorizadas()
        return  usuario  ==  usuario_autorizado  and  clave  ==
clave_autorizada

def credenciales_autorizadas():
    usuario_autorizado=""
    clave_autorizada=""
            with   open("./repositorios/usuario.csv","r")   as
credenciales_csv:

            lector_csv = csv.DictReader(credenciales_csv)

            for fila in lector_csv:
                usuario_autorizado = fila["usuario"]
                clave_autorizada = fila["clave"]
    return [usuario_autorizado,clave_autorizada]
```

2. **Diagrama de flujo**



3. Grafo de Flujo (GF)



4. **Identificación de rutas**
   R1: 1,2,4,5
   R2: 1,2,3,5
5. **Complejidad Ciclomática**
   Se puede calcular de las siguientes formas:

- V(G) = número de nodos predicados(decisiones)+1
  V(G)= 2+1=3

- V(G) = A – N + 2
  V(G)= 5−5+2=2

DONDE:
P: Número de nodos predicado
A: Número de aristas
N: Número de nodos

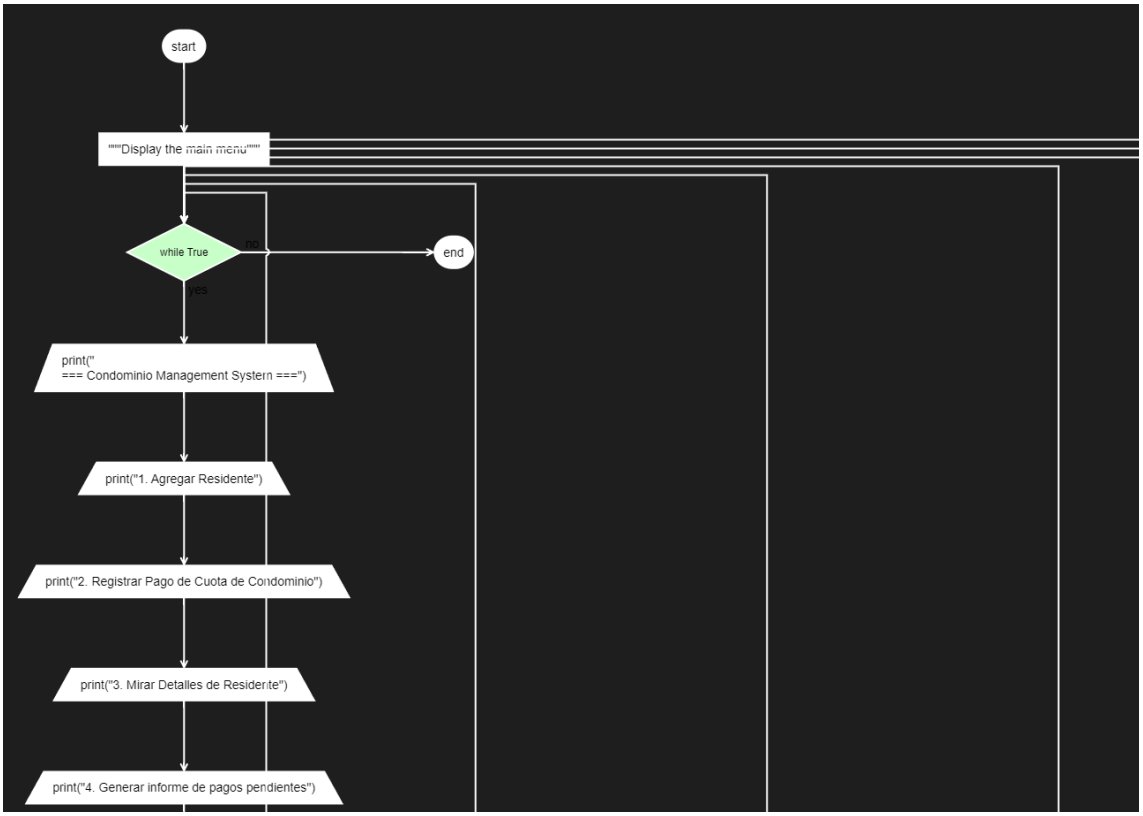
Prueba de caja blanca de RF 2.menú

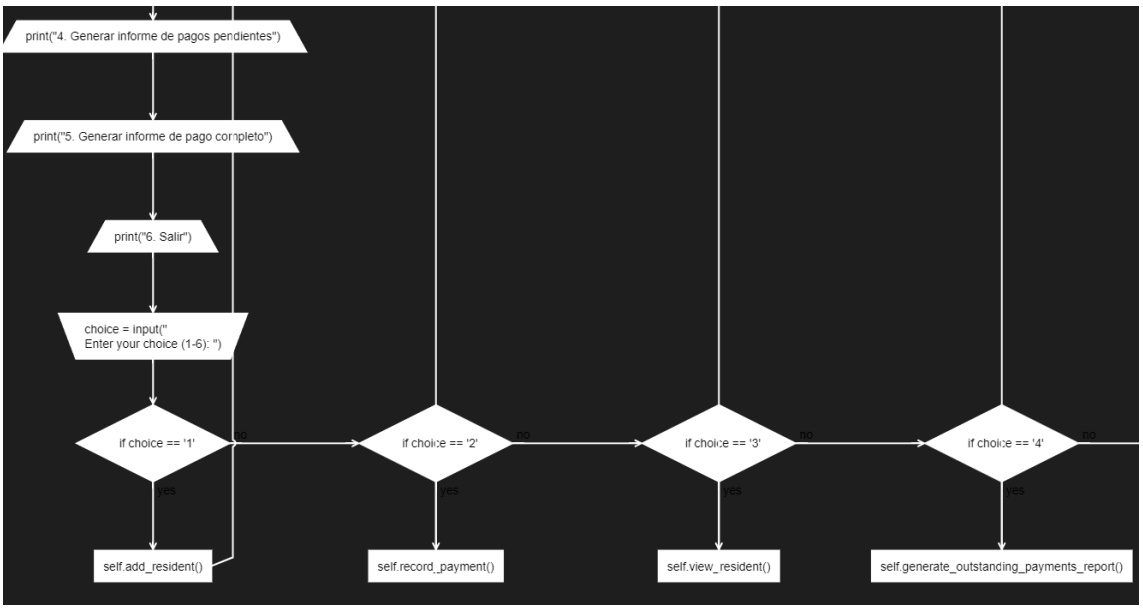
1. Código fuente

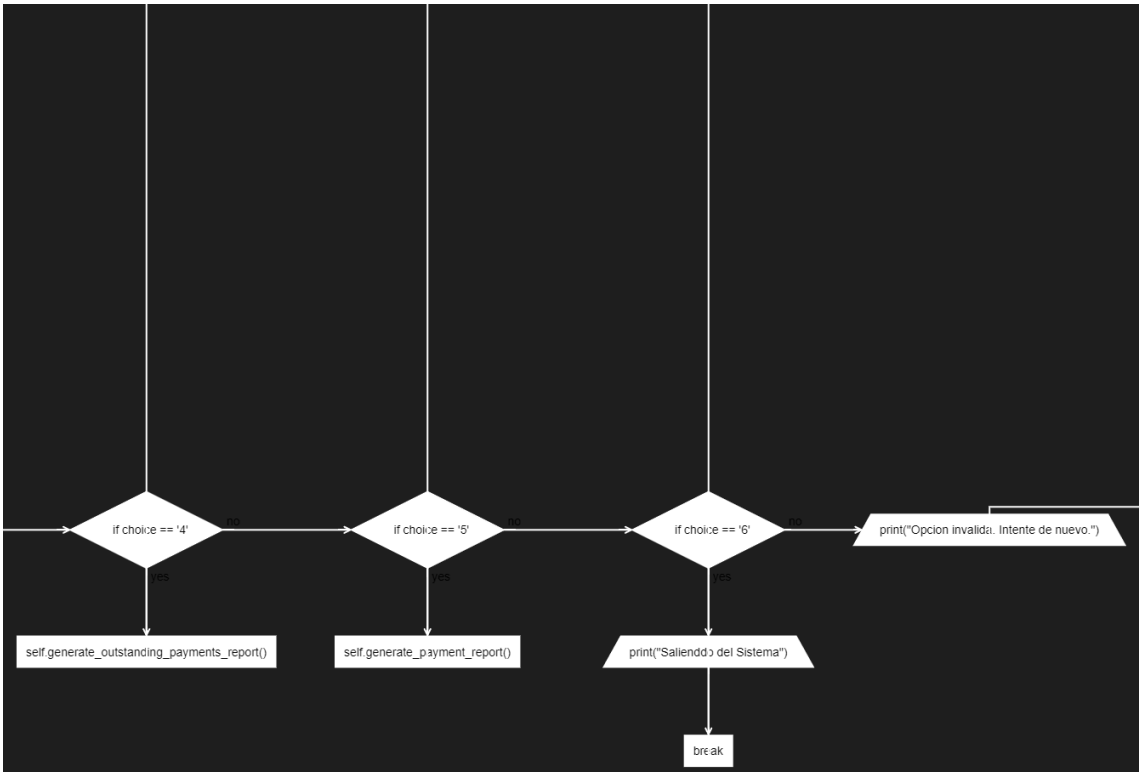
```
def display_menu(self):
        """Display the main menu"""
        while True:
            print("\n=== Condominio Management System ===")
            print("1. Agregar Residente")
            print("2. Registrar Pago de Cuota de Condominio")
            print("3. Mirar Detalles de Residente")
            print("4. Generar informe de pagos pendientes")
            print("5. Generar informe de pago completo")
```

```
            print("6. Salir")

            choice = input("\nEnter your choice (1-6): ")

            if choice == '1':
                self.add_resident()
            elif choice == '2':
                self.record_payment()
            elif choice == '3':
                self.view_resident()
            elif choice == '4':
                self.generate_outstanding_payments_report()
            elif choice == '5':
                self.generate_payment_report()
            elif choice == '6':
                print("Salienddo del Sistema")
                break
            else:
                print("Opcion invalida. Intente de nuevo.")
```

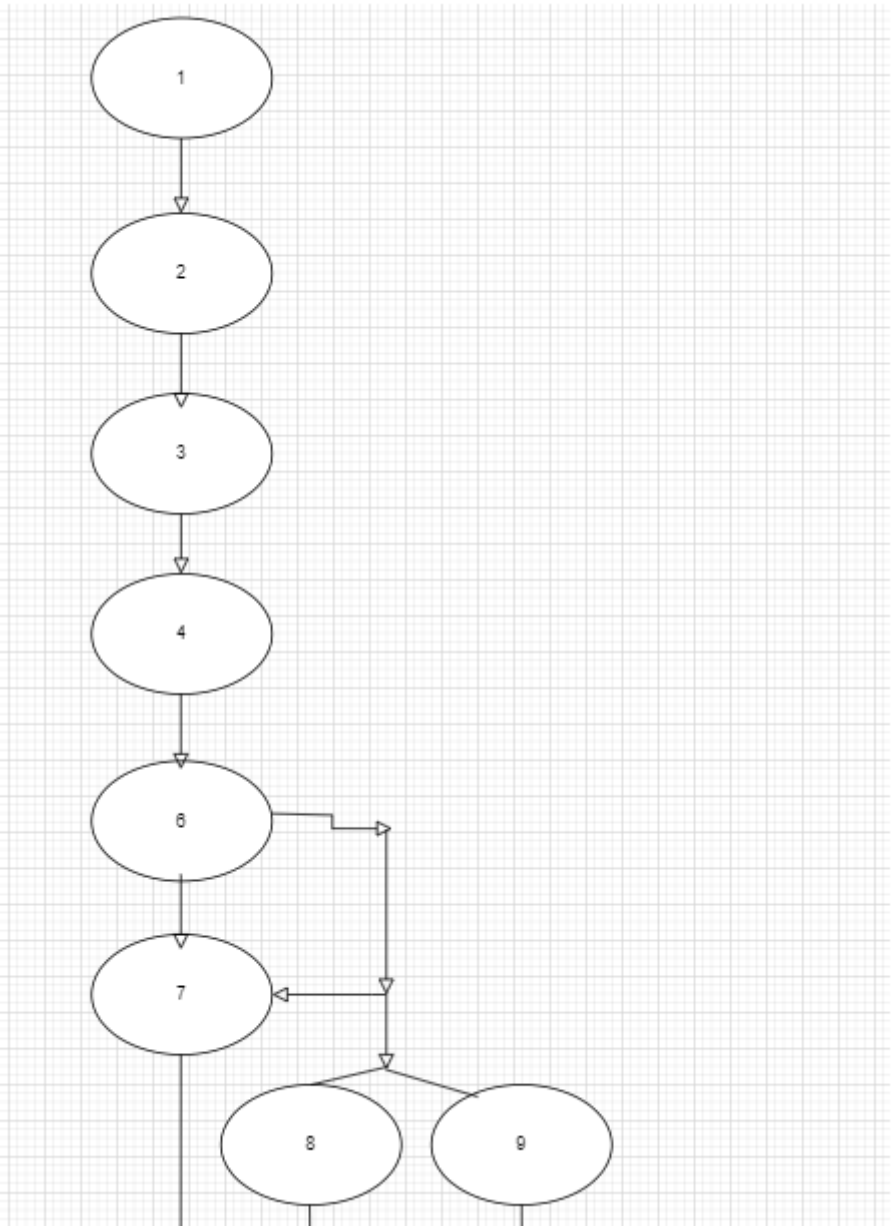
2. Diagrama de flujo

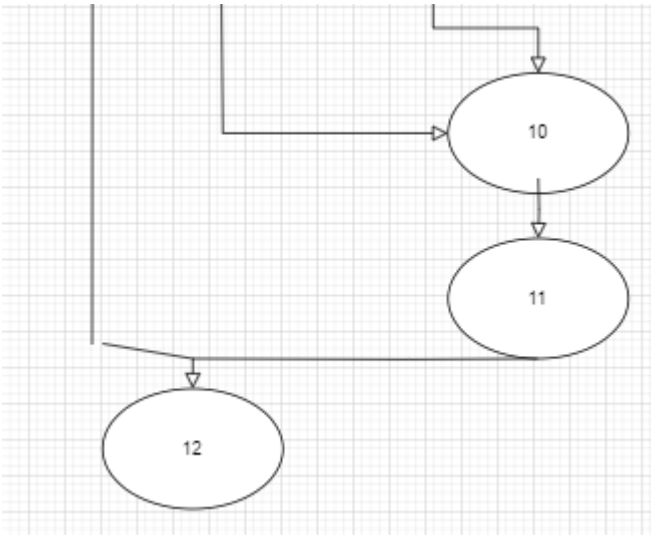

start
"""Display the main menu"""
while True
no
end
yes
print("
=== Condominio Management System ===")
print("1. Agregar Residente")
print("2. Registrar Pago de Cuota de Condominio")
print("3. Mirar Detalles de Residente")
print("4. Generar informe de pagos pendientes")


print("4. Generar informe de pagos pendientes")
print("5. Generar informe de pago completo")
print("6. Salir")
choice = input("
Enter your choice (1-6): ")
if choice == '1'
no
yes
self.add_resident()
if choice == '2'
no
yes
self.record_payment()
if choice == '3'
no
yes
self.view_resident()
if choice == '4'
no
yes
self.generate_outstanding_payments_report()



3. Grafo de flujo



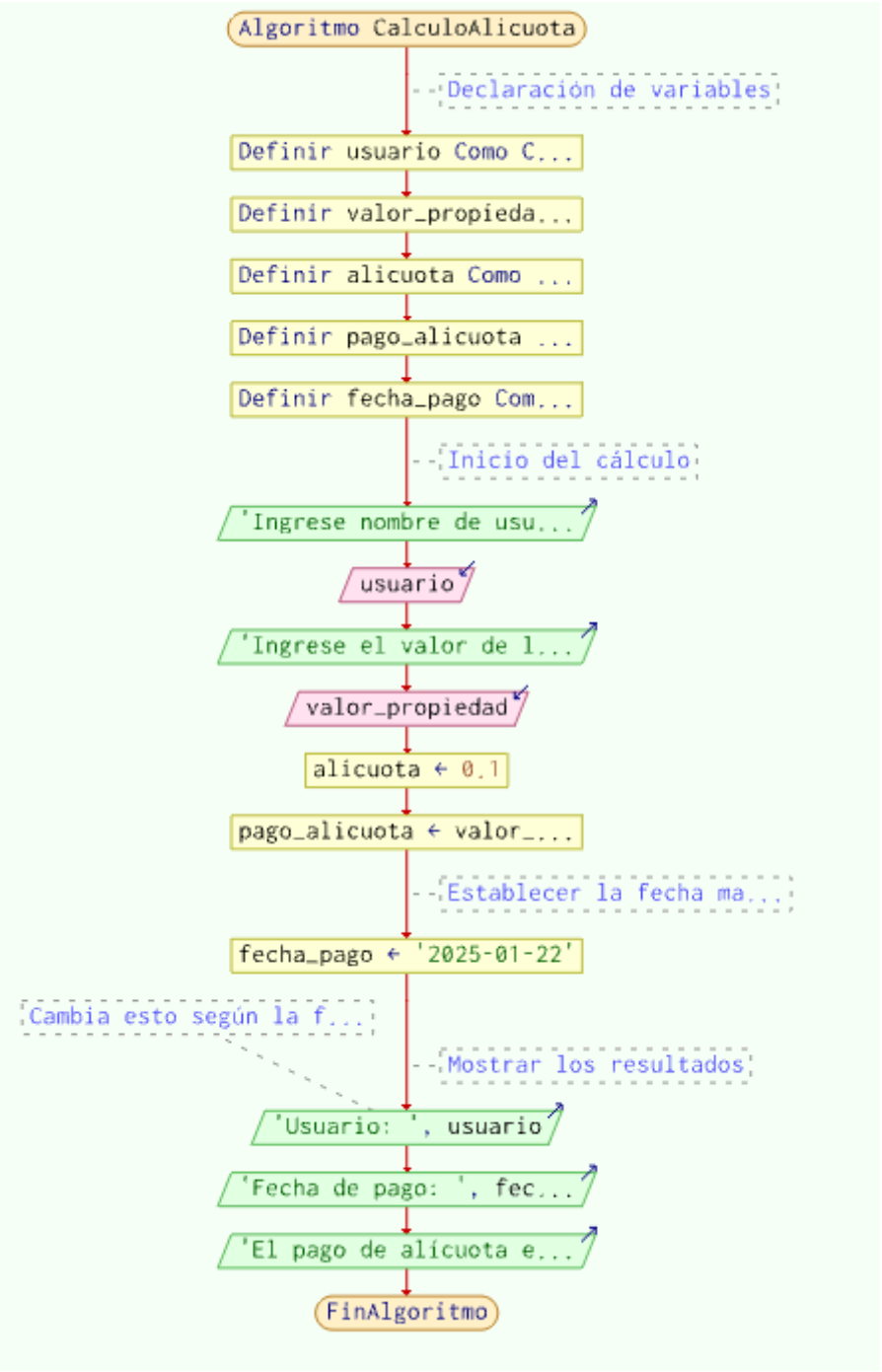


Prueba caja blanca de RF 3. Notificación de pagos pendientes

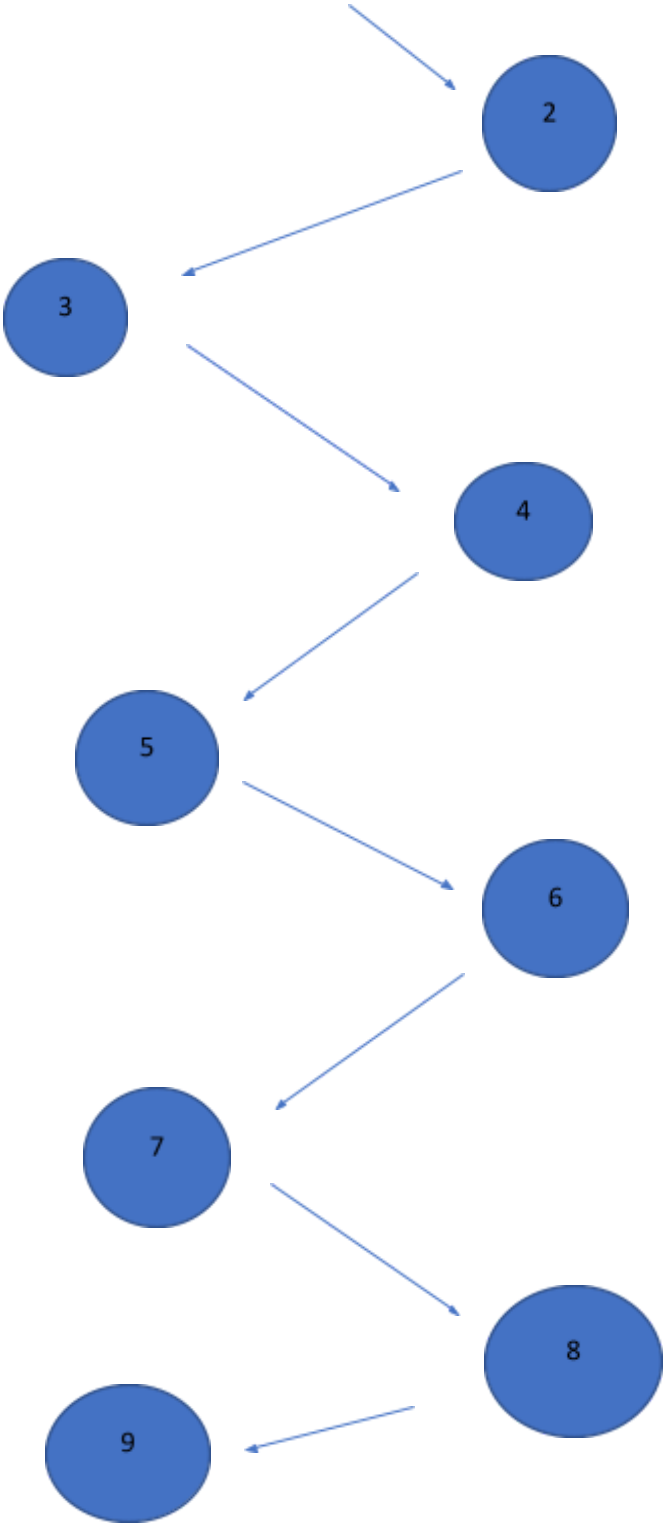
1. CÓDIGO FUENTE

```python
import json
from datetime import datetime

# Archivo para guardar pagos pendientes
ARCHIVO_PAGOS = 'pagos_pendientes.json'

# Cargar datos de archivos
def cargar_datos(archivo):
    try:
        with open(archivo, 'r') as f:
            return json.load(f)
    except FileNotFoundError:
        return {}

def guardar_datos(archivo, datos):
    with open(archivo, 'w') as f:
        json.dump(datos, f)

# Función para calcular el pago de alícuota
pagos_pendientes = cargar_datos(ARCHIVO_PAGOS)

def calcular_alicuota():
    usuario = input("Ingrese nombre de usuario: ")
    valor_propiedad = float(input("Ingrese el valor de la propiedad: "))
    alicuota = 0.1
    pago_alicuota = valor_propiedad * alicuota
    fecha_pago = datetime.now()
    pagos_pendientes[usuario] = fecha_pago.strftime('%Y-%m-%d')
    guardar_datos(ARCHIVO_PAGOS, pagos_pendientes)
    print(f"El pago de alícuota es: {pago_alicuota:.2f}")

if __name__ == "__main__":
    calcular_alicuota()

```

2.DIAGRAMA DE FLUJO (DF)



3. GRAFO DE FLUJO (GF)



4. **IDENTIFIACCIÒN DE LAS RUTAS (Camino basico)**

Ruta 1: Nodo 1 → Nodo 2 → Nodo 3 → Nodo 4 → Nodo 5 → Nodo 6 → Nodo7 → Nodo 8 → Nodo 9

5. **Complejidad ciclomática**

CC=E−N+2PCC = E - N + 2PCC=E−N+2P

- EEE = Número de aristas (líneas entre nodos en el grafo).
- NNN = Número de nodos.
- PPP = Número de componentes conexos (normalmente 1 para algoritmos simples).

- E=8E = 8E=8 (hay 8 líneas de conexión entre los nodos).
- N=9N = 9N=9 (hay 9 nodos en el grafo).
- P=1P = 1P=1 (todo está en un solo componente).

CC=8−9+2(1)=1CC = 8 - 9 + 2(1) = 1CC=8−9+2(1)=1

La **complejidad ciclomática** es:

CC=1CC = 1CC=1

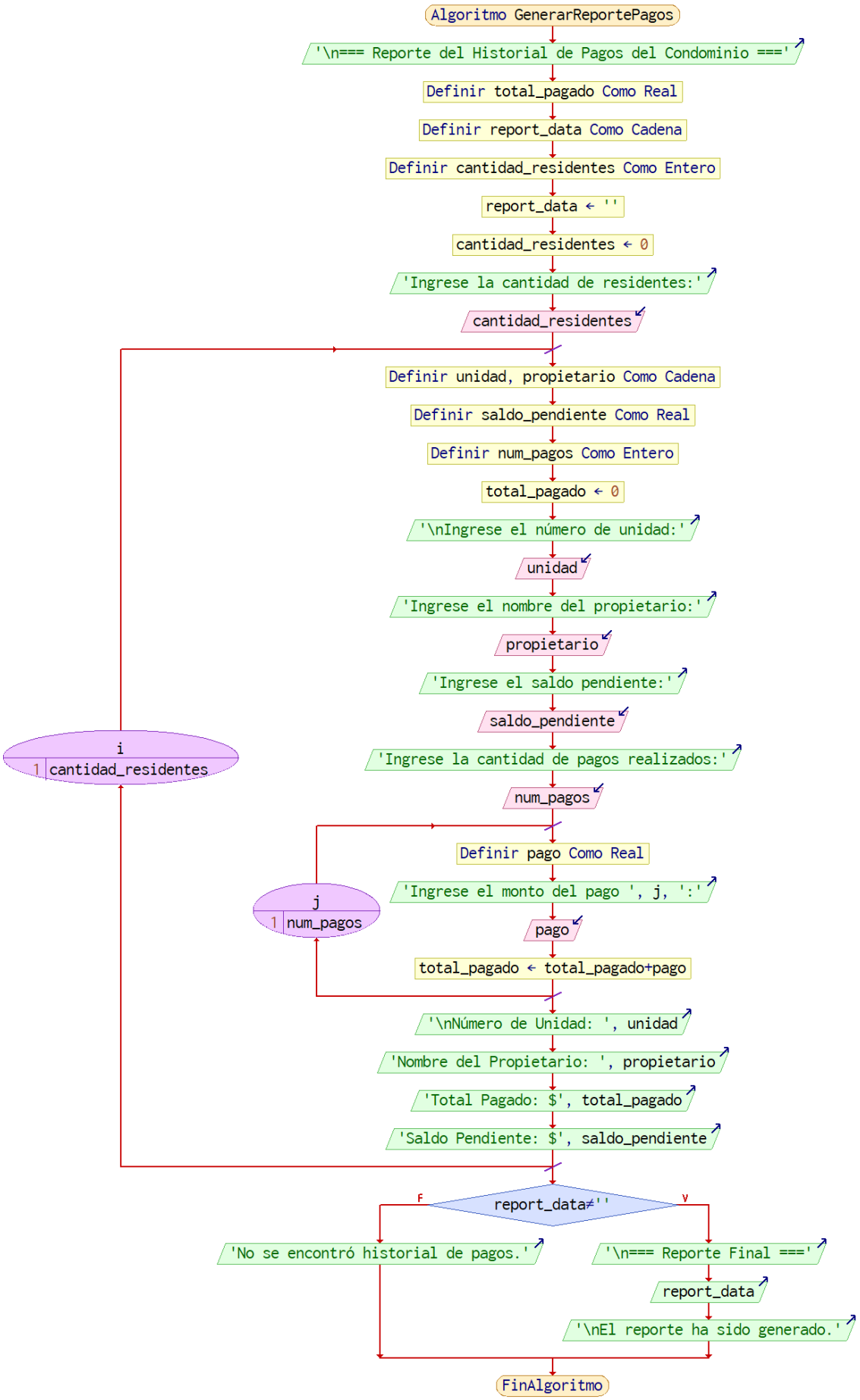
Esto confirma que el algoritmo tiene solo un camino básico, sin bifurcaciones.

Prueba caja blanca de RF 5.Generación de reportes

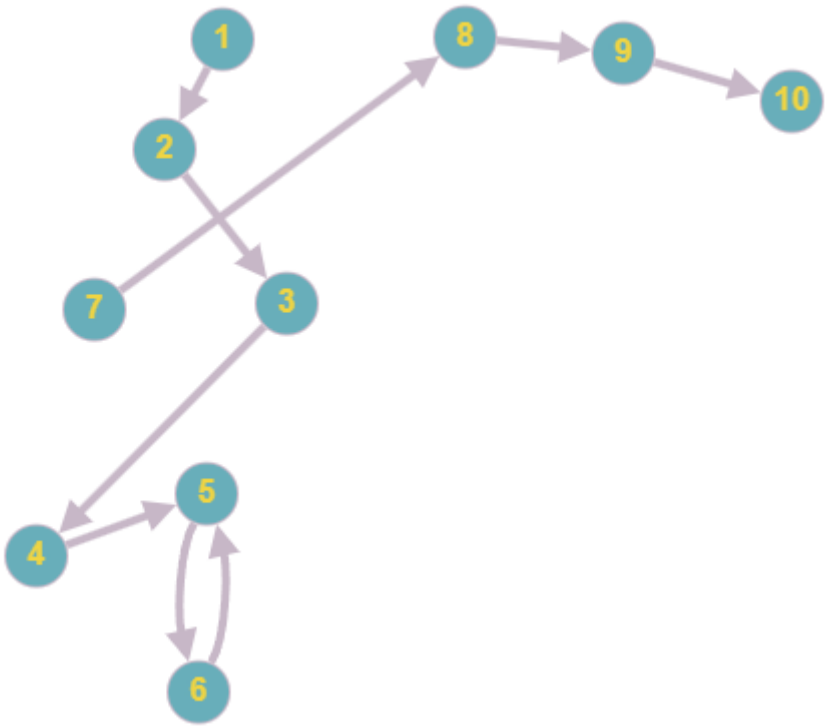
1. **CÓDIGO FUENTE**

```
def generate_payment_report(self):
    """Generar el historial completo de pagos de la cuota del condominio"""
    print("\n=== Reporte del Historial de Pagos del Condominio ===")
    report_data = []
     headers = ['Número de Unidad', 'Nombre del Propietario', 'Total Pagado', 'Saldo Pendiente']

    for unit_number, data in self.residents.items():
        total_paid = sum(payment['amount'] for payment in data['payment_history'])
        report_data.append({
            'Número de Unidad': unit_number,
            'Nombre del Propietario': data['owner_name'],
            'Total Pagado': f"${total_paid:.2f}",
            'Saldo Pendiente': f"${data['outstanding_balance']:.2f}"
        })
        print(f"\nNúmero de Unidad: {unit_number}")
        print(f"Nombre del Propietario: {data['owner_name']}")
        print(f"Total Pagado: ${total_paid:.2f}")
        print(f"Saldo Pendiente: ${data['outstanding_balance']:.2f}")

    if report_data:
        self.generate_report_file(report_data, 'payment_history_report.csv', headers)
        print("\nReporte guardado en 'payment_history_report.csv'")
    else:
        print("No se encontró historial de pagos.")
```

2. **DIAGRAMA DE FLUJO (DF)**


Algoritmo GenerarReportePagos
'\n=== Reporte del Historial de Pagos del Condominio ==='
Definir total_pagado Como Real
Definir report_data Como Cadena
Definir cantidad_residentes Como Entero
report_data ← ''
cantidad_residentes ← 0
'Ingrese la cantidad de residentes:'
cantidad_residentes
i
1 cantidad_residentes
Definir unidad, propietario Como Cadena
Definir saldo_pendiente Como Real
Definir num_pagos Como Entero
total_pagado ← 0
'\nIngrese el número de unidad:'
unidad
'Ingrese el nombre del propietario:'
propietario
'Ingrese el saldo pendiente:'
saldo_pendiente
'Ingrese la cantidad de pagos realizados:'
num_pagos
j
1 num_pagos
Definir pago Como Real
'Ingrese el monto del pago ', j, ':'
pago
total_pagado ← total_pagado+pago
'\nNúmero de Unidad: ', unidad
'Nombre del Propietario: ', propietario
'Total Pagado: $', total_pagado
'Saldo Pendiente: $', saldo_pendiente
report_data≠''
F
V
'No se encontró historial de pagos.'
'\n=== Reporte Final ==='
report_data
'\nEl reporte ha sido generado.'
FinAlgoritmo

3. GRAFO DE FLUJO (GF)



4. IDENTIFIACCIÒN DE LAS RUTAS (Camino basico)

Determinar en base al GF del numeral 4
RUTAS

R1: 1,2,7,3,8,9,10
R2: 1,2,3,4,5,6,5,3,8,9,10

5. COMPLEJIDAD CICLOMÁTICA

Se puede calcular de las siguientes formas:

- V(G) = número de nodos predicados(decisiones)+1
  V(G)= 2+1=3

- V(G) = A – N + 2
  V(G)= 12−10+2=3

DONDE:
P: Número de nodos predicado
A: Número de aristas
N: Número de nodos