# Pruebas de Caja Blanca

***"Sistema de pagos de alícuotas"***

**Integrantes:**
**Lucas Góngora**
**Gabriel Manosalvas**
**Jairo Molina**
**Yandry Vélez**

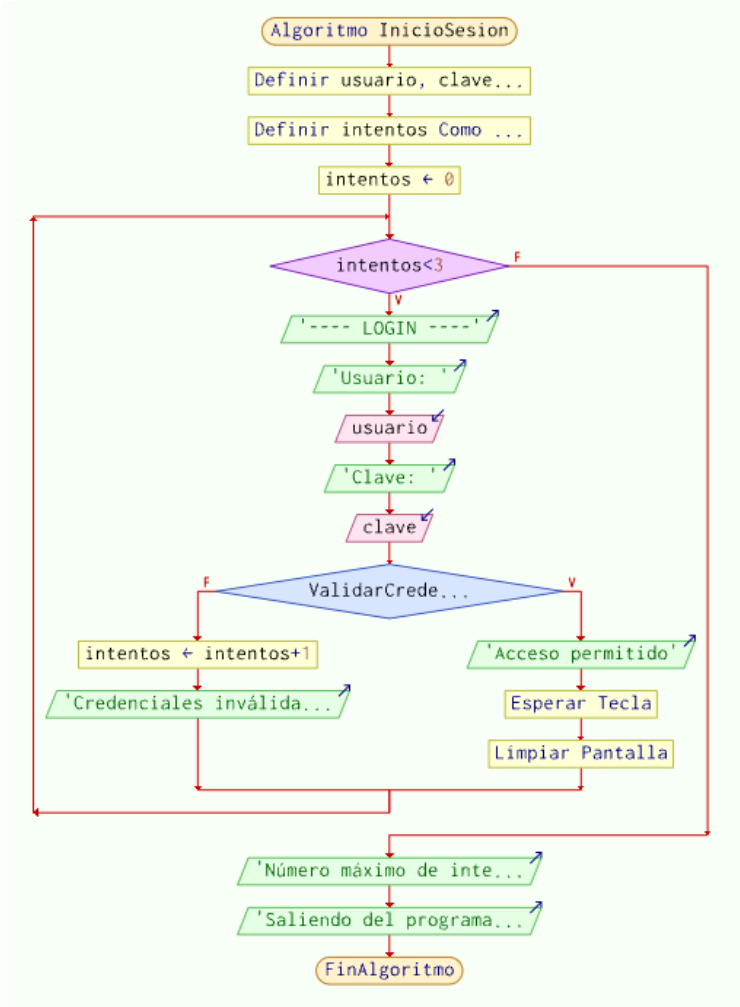
**Prueba caja blanca de** RF 1.Login

1. **Código FUENTE**

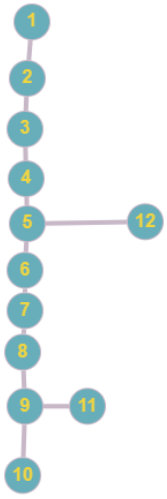
```
import re
import getpass
from csv_managment import *
from consola import *


validacion_username = r'^[a-zA-Z]+$'
validacion_clave = r'^(?=.*[!@#$%^&*(),.?":{}|<>])(?=.{8,})'


def iniciar_sesion():
    limpiar_pantalla()

    clave_incorrecta = True
    intentos_realizados = 0
    intentos_permitidos = 3

    if obtener_credenciales_autorizadas() == []:
        registrar_administrador()

    while clave_incorrecta:
        print("\t----LOGIN---\t")

        usuario, clave = obtener_credenciales()
        validacion_exitosa = validar_credenciales(usuario, clave)

        if validacion_exitosa:
            clave_incorrecta = False
            print("\nCredenciales válidas")
            input("Presiona ENTER para pasar al menú ...")
            limpiar_pantalla()

        else:
            print("\nCredenciales inválidas\n")
            intentos_realizados += 1
            print("Te quedan " + str(intentos_permitidos -
intentos_realizados) + " intentos")
            input("Presiona ENTER para volver al Login ...")
            limpiar_pantalla()

            if intentos_realizados >= intentos_permitidos:
                print("Número de intentos permitidos alcanzados")
                salir_programa()


def registrar_administrador():
    print("\tREGISTRAR\t")
    credenciales = obtener_credenciales()
```

```
    insertar("./databases/administrador.csv", credenciales,
"./databases")
    limpiar_pantalla()


def validar_credenciales(usuario, clave):
    usuario_autorizado, clave_autorizada =
obtener_credenciales_autorizadas()[0]
    return usuario == usuario_autorizado and clave == clave_autorizada


def obtener_credenciales_autorizadas():
    credenciales_autorizadas = leer("./databases/administrador.csv")
    return credenciales_autorizadas


def obtener_credenciales():
    usuario = validar_entrada("username", validacion_username,
"Ingresa solo letras")
    clave = validar_entrada("clave", validacion_clave, "Ingresa al
menos 8 caracteres y al menos un caracter especial")
    return [usuario, clave]


def validar_entrada(tipo, patron, mensaje_error):
    while True:
        if tipo == "clave":
            entrada = getpass.getpass("Ingresa la clave: ")
        elif tipo == "username":
            entrada = input("Ingresa el nombre de usuario: ")
        else:
            print("Ingresa un tipo de entrada válido")
            break
        if re.match(patron, entrada):
            return entrada
   print(mensaje_error)
```

## 2. Diagrama de flujo




3. **Grafo de Flujo (GF)**




4. **Identificación de rutas**
   R1: 1,2,4,5,6,7,8,9,10.
   R2: 1,2,3,5,12.
5. **Complejidad Ciclomática**

**P=2**
**N=12**
**A=14**

Se puede calcular de las siguientes formas:

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

- V(G) = A – N + 2
  V(G)=14−12+2=4

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

**Prueba de caja blanca de RF 2.menú**

```python
from consola import *
from alicuota import *

def mostrar_menu():
    while True:
        opciones_menu()
        opcion = obtener_opcion()
        if opcion == 0:
            continue
        ejecutar_opcion(opcion)

def opciones_menu():
    print("\t------MENU------\t")
    print("1. Registrar alicuota")
    print("2. Mostrar alicuotas")
    print("3. Buscar alicuota")
    print("4. Actualizar alicuota")
    print("5. Mostrar alicuotas pendientes por pagar")
    print("6. Reporte de alicuotas en excel")
    print("7. Salir del programa")

def obtener_opcion():
    option = int(input("Ingresa tu opcion: "))
    if 0 < option < 8:
        return option
    else:
        print("Ingresa una opcion valida")
        input("Presiona ENTER para volver al menu")
        limpiar_pantalla()
        return 0

def ejecutar_opcion(opcion):
    opciones = {
        1: opt_registrar_alicuota,
        2: opt_mostrar_alicuotas,
        3: opt_buscar_alicuota,
        4: opt_actualizar_alicuota,
```

```
        5: opt_mostrar_alicuotas_pendientes,
        6: opt_reporte_alicuotas,
        7: salir_programa
    }
    if opcion in opciones:
        opciones[opcion]()


def opt_registrar_alicuota():
    print("Registrar una alícuota:")
    datos = solicitar_datos_alicuota()
    if datos:
        registrar_alicuota(*datos)
        print("Alícuota registrada correctamente.")
    input("Presiona ENTER PARA VOLVER AL MENU PRINCIPAL")
    limpiar_pantalla()

def opt_actualizar_alicuota():
    print("Actualizar una alícuota:")

    id = input("ID de la alícuota: ")

    alicuota_actual = buscar_por_id("./databases/alicuota.csv", id)

    if alicuota_actual:
        print("Datos actuales de la alícuota:")
        print(alicuota_actual)
        datos = solicitar_datos_alicuota()
        if datos:
            actualizar_alicuota(id, *datos)
            print("Alícuota actualizada correctamente.")
    else:
        print("Alícuota no encontrada.")
    input("Presiona ENTER PARA VOLVER AL MENU PRINCIPAL")
    limpiar_pantalla()
```

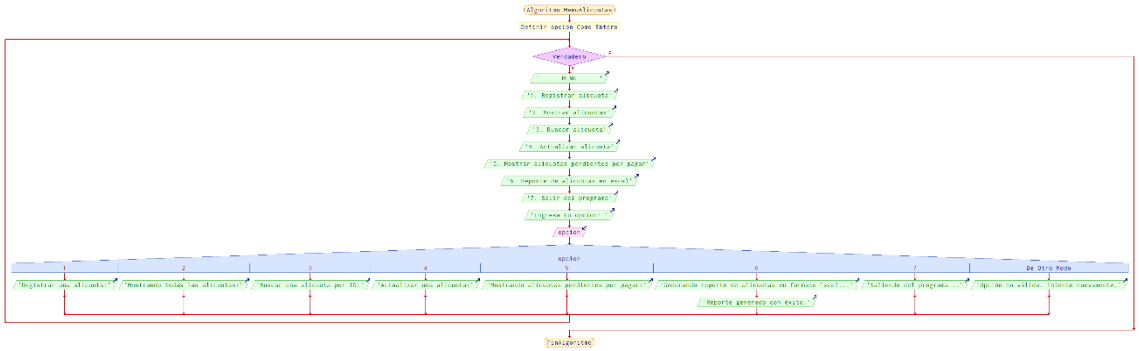
C:\Users\Gabriel\Downloads\codeFundamentos\codeFundamentos\menu.py

```python
def opt_mostrar_alicuotas():
    print("Mostrando todas las alícuotas:")
    mostrar_alicuotas()
    input("Presiona ENTER PARA VOLVER AL MENU PRINCIPAL")
    limpiar_pantalla()

def opt_buscar_alicuota():
    print("Buscar una alícuota por ID:")
    id = input("ID de la alícuota: ")
    buscar_alicuota(id)
    input("Presiona ENTER PARA VOLVER AL MENU PRINCIPAL")
    limpiar_pantalla()

def opt_mostrar_alicuotas_pendientes():
    print("Mostrando alícuotas pendientes por pagar:")
    buscar_alicuotas_pendientes()
    input("Presiona ENTER PARA VOLVER AL MENU PRINCIPAL")
    limpiar_pantalla()

def opt_reporte_alicuotas():
    print("Generando reporte de alícuotas en formato Excel...")

    print("Reporte generado con éxito.")

def salir_programa():
    print("Saliendo del programa...")
    exit()

def solicitar_datos_alicuota():
    id = input("ID de la alícuota: ")
    residente = input("Nombre del residente: ")

    estado_pago = input("Estado de pago (pendiente/pagado): ")

    if estado_pago not in ["pendiente", "pagado"]:
        print("El estado de pago debe ser 'pendiente' o 'pagado'.")
        return None
```

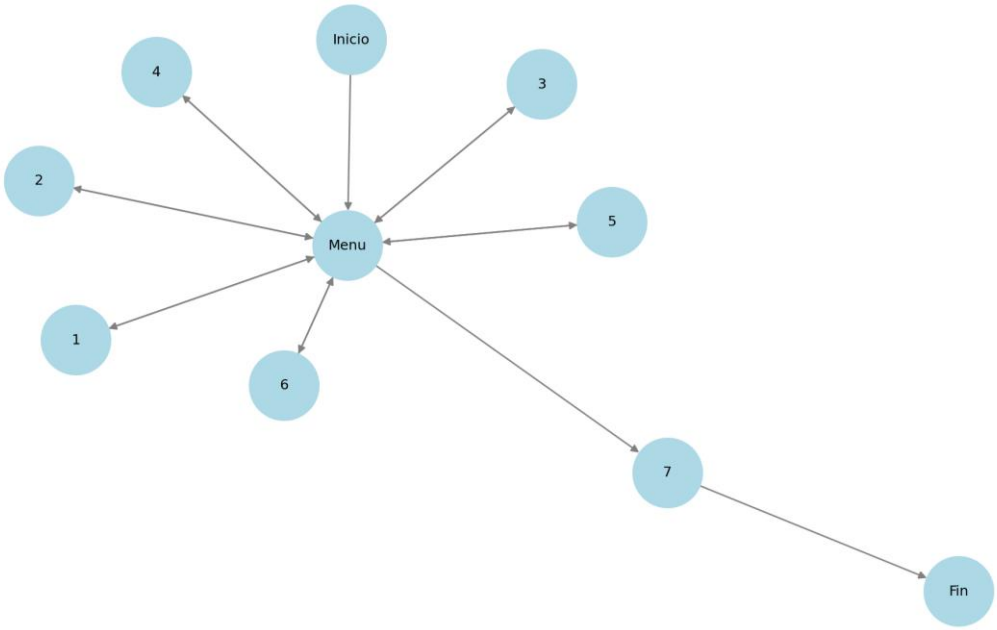
```
    porcentaje_alicuota = input("Porcentaje de alicuota: ")

    if not es_numero_valido(porcentaje_alicuota):
        return None

    porcentaje_alicuota = float(porcentaje_alicuota)

    base_imponible = input("Base imponible: ")

    if not es_numero_valido(base_imponible):
        return None

    base_imponible = float(base_imponible)

    descripcion = input("Descripción: ")
    multa = input("Monto de la multa: ")

    if not es_numero_valido(multa):
        return None
    multa = float(multa)

    descuento = input("Monto de descuento: ")
    if not es_numero_valido(descuento):
        return None
    descuento = float(descuento)

    ajuste_extraordinario = input("Ajuste extraordinario: ")
    if not es_numero_valido(ajuste_extraordinario):
        return None
    ajuste_extraordinario = float(ajuste_extraordinario)

    return id, residente, estado_pago, porcentaje_alicuota, base_imponible, descripcion, multa, descuento, ajuste_extraordinario
```

```
def es_numero_valido(cadena):
    if not cadena.replace(".", "", 1).isdigit() or cadena.count('.') > 1:
        print(f"Por favor, ingresa un numero válido.")
        return False
    return True
```

2. Diagrama de flujo



3. Grafo de flujo




## 4. Identificación de rutas

**a. R1:** 1,2,3,4,6,7.
**b. R2:** 1,2,3,4,6,5,7.
**c. R3:** 1,2,3,4,6,3,7.
**d. R4:** 1,2,3,4,6,4,7.
**e. R5:** 1,2,3,4,6,2,7.
**f. R6:** 1,2,3,4,6,1,7.

**Complejidad Ciclomática Se puede calcular de las siguientes formas:**

**N= 7**
**A=12**
**P=5**

**V(G)=P+1**

**V(G)=5+1=6**

**V(G)=A−N+2**

V(G)=12−7+2=7

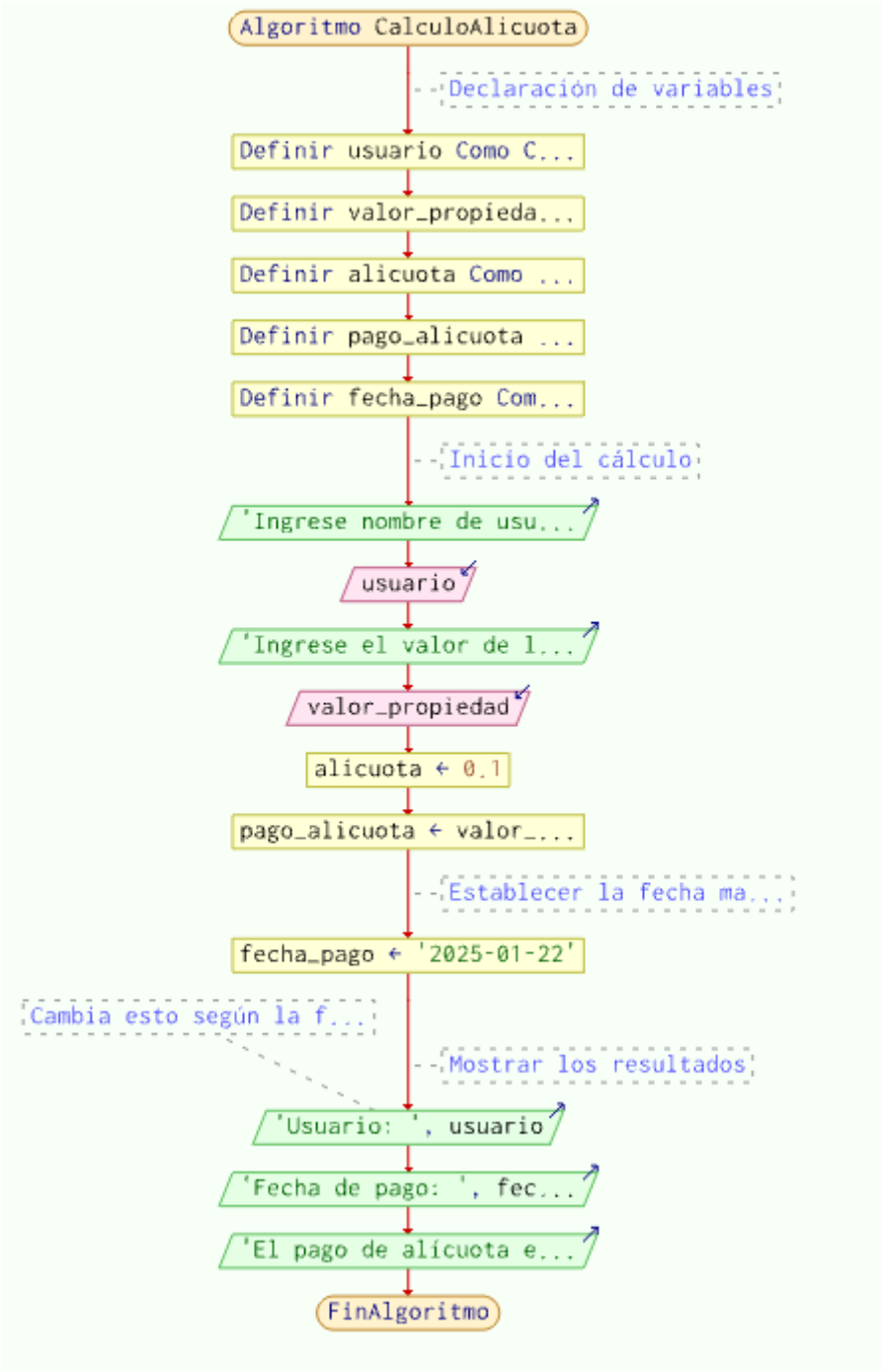
**DONDE:**
**P: Número de nodos predicado**
**A: Número de aristas**
**N: Número de nodos**

**Prueba caja blanca de RF 3. Notificación de pagos pendientes**

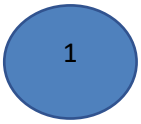
1. CÓDIGO FUENTE

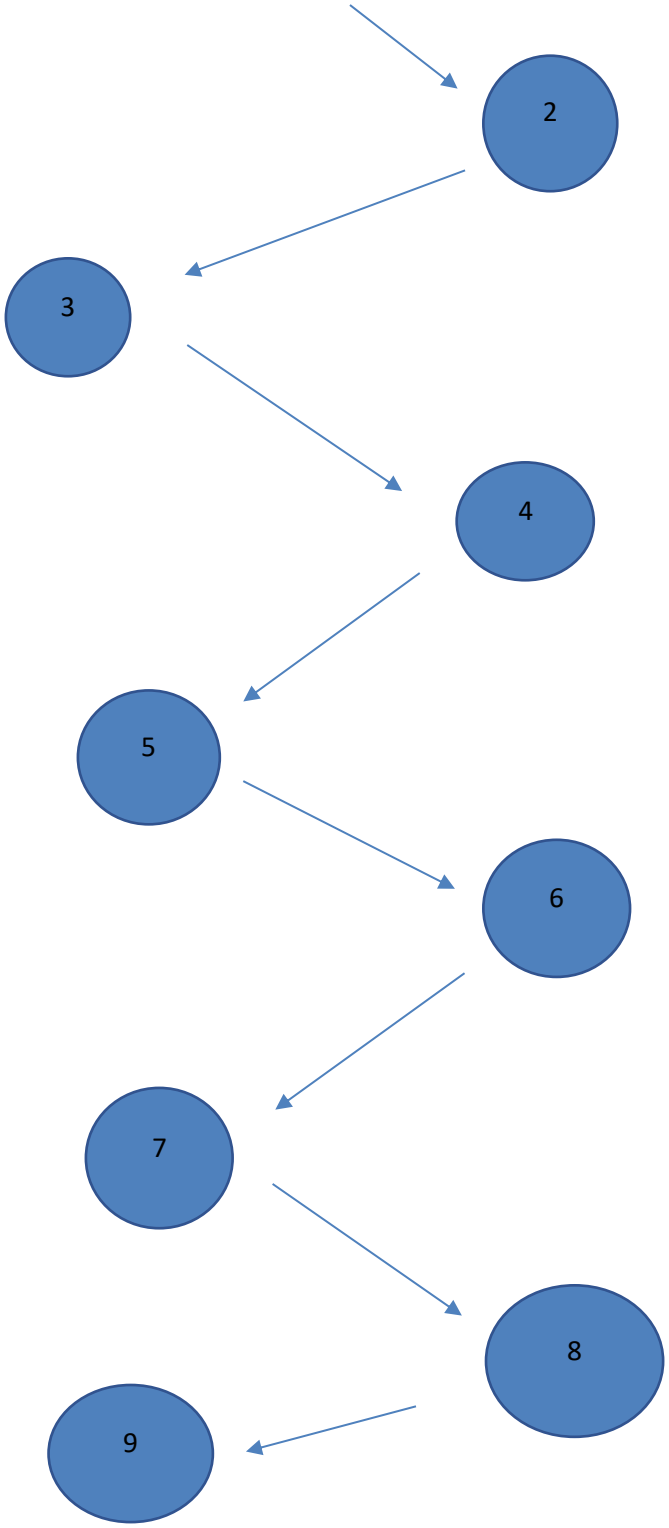
```python
import json
from datetime import datetime

# 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

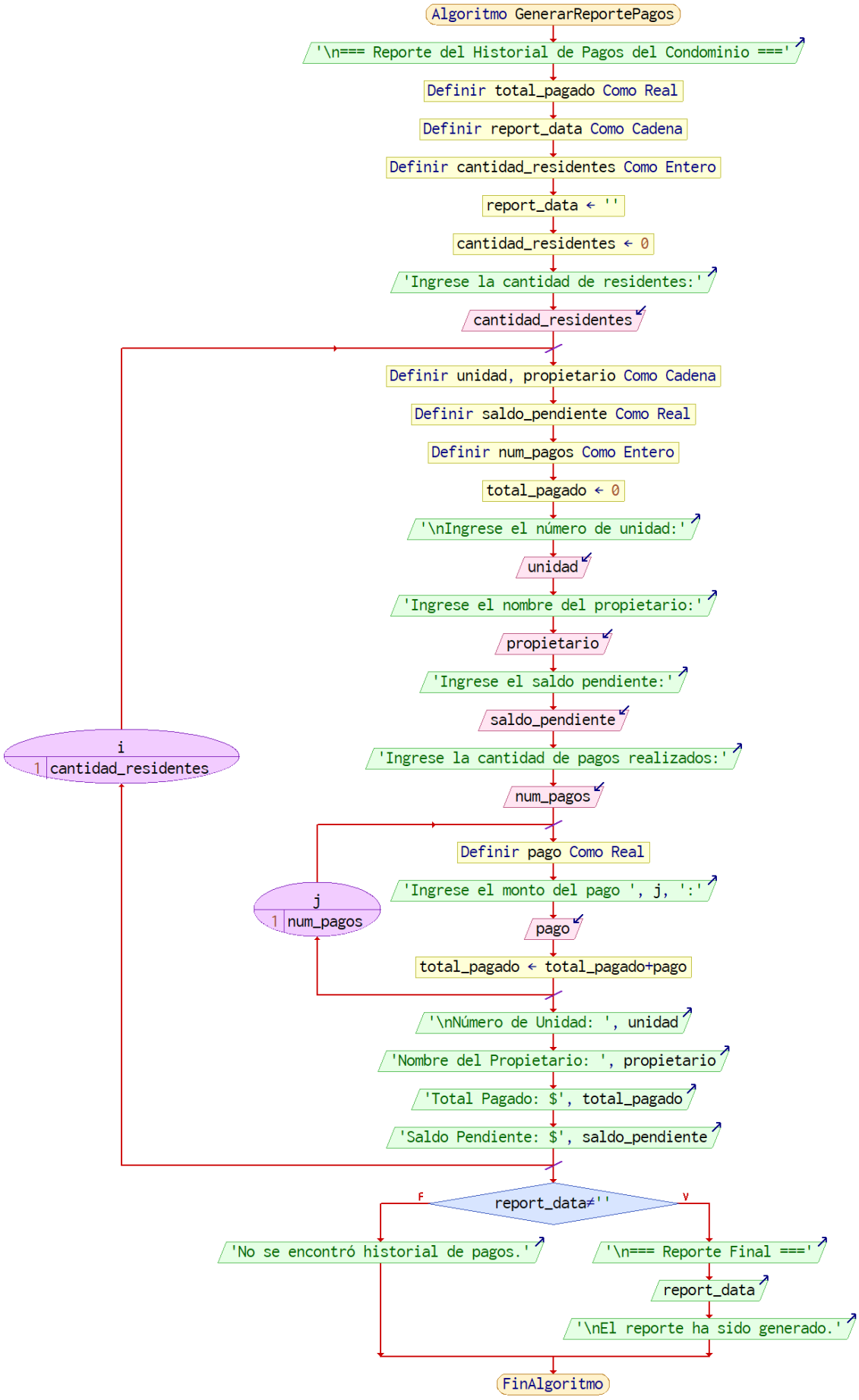
Esto confirma que el algoritmo tiene solo un camino básico, sin bifurcaciones.

**Prueba caja blanca de** RF 5.Generación de reportes

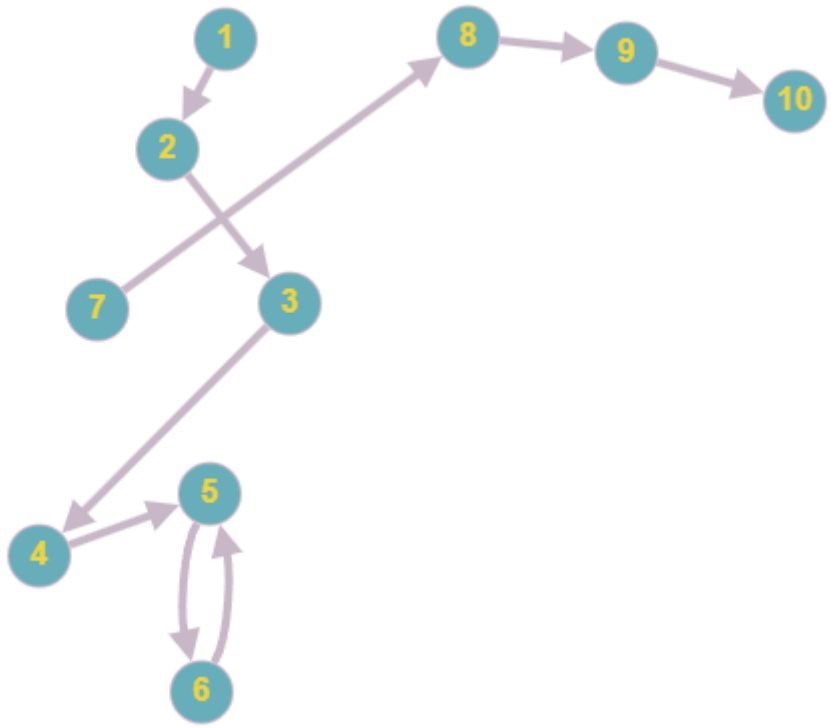
1. **CÓDIGO FUENTE**

```
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