Prueba de Caja Blanca

"Sistema computarizado de registro de ingreso y salida vehicular y peatonal para el conjunto habitacional Armenia Etapa II"

Integrantes:
Kerly Andreina Chuqui Aguinda
Denis Alexander Ullcu Ullco

Fecha 2024-08-06

Prueba caja blanca de Requisito 1: Registro de ingreso de usuarios, existentes y nuevos.

1. **CÓDIGO FUENTE**

```
#include <iostream>
#include <string>
#include <vector>
#include <ctime>
#include <iomanip>
#include <regex>
#include <fstream>
#include <cstdlib> // Para usar system()

using namespace std;

struct Usuario {
    string nombreCompleto;
    string numeroCasa;
    string numeroCedula;
    string horaEntrada;
    string horaSalida;
    string placaVehicular;
    string tipoUsuario; // "Residente" o "Visita"
};

vector<Usuario> usuarios;

string obtenerHoraActual() {
    time_t now = time(0);
    tm *ltm = localtime(&now);
    ostringstream oss;
    oss << setw(2) << setfill('0') << ltm->tm_hour << ":"
        << setw(2) << setfill('0') << ltm->tm_min << ":"
        << setw(2) << setfill('0') << ltm->tm_sec;
    return oss.str();
}

bool validarCedula(const string &cedula) {
    regex cedulaRegex("\\d{10}");
    return regex_match(cedula, cedulaRegex);
}

// Función para guardar usuarios en un archivo de texto
void guardarUsuariosEnArchivo() {
    ofstream archivo("usuarios.txt");
    if (!archivo.is_open()) {
        cerr << "Error al abrir el archivo para escribir.\n";
        return;
    }

    for (const auto &usuario : usuarios) {
        archivo << "Nombre: " << usuario.nombreCompleto << "\n"
                << "Numero de casa: " << usuario.numeroCasa << "\n"
                << "Numero de cedula: " << usuario.numeroCedula << "\n"
                << "Hora de entrada: " << usuario.horaEntrada << "\n"
                << "Hora de salida: " << usuario.horaSalida << "\n"
                << "Placa vehicular: " << usuario.placaVehicular << "\n"
                << "Tipo de usuario: " << usuario.tipoUsuario << "\n"
                << "----------------------------------------\n";
    }
    archivo.close();
```

```
}

void agregarUsuario() {
    Usuario nuevoUsuario;
    cout << "Ingrese numero de cedula (10 digitos): ";
    getline(cin, nuevoUsuario.numeroCedula);

    if (!validarCedula(nuevoUsuario.numeroCedula)) {
        cout << "Cedula invalida. Debe contener exactamente 10 digitos.\n";
        return;
    }

    cout << "Ingrese nombre completo: ";
    getline(cin, nuevoUsuario.nombreCompleto);
    cout << "Es residente o visita? (r/v): ";
    char tipoUsuario;
    cin >> tipoUsuario;
    cin.ignore();

    if (tipoUsuario == 'r' || tipoUsuario == 'R') {
        nuevoUsuario.tipoUsuario = "Residente";
        cout << "Ingrese numero de casa: ";
        getline(cin, nuevoUsuario.numeroCasa);
        nuevoUsuario.horaSalida = obtenerHoraActual();
        nuevoUsuario.horaEntrada = "N/A";
          cout << "Usuario agregado exitosamente. Hora de Salida: " << nuevoUsuario.horaSalida
<< "\n";

    } else if (tipoUsuario == 'v' || tipoUsuario == 'V') {
        nuevoUsuario.tipoUsuario = "Visita";
        cout << "Ingrese numero de casa: ";
        getline(cin, nuevoUsuario.numeroCasa);
        nuevoUsuario.horaEntrada = obtenerHoraActual();
        nuevoUsuario.horaSalida = "N/A";
                    cout << "Usuario agregado exitosamente. Hora de entrada: " <<
nuevoUsuario.horaEntrada << "\n";
    } else {
        cout << "Tipo de usuario invalido.\n";
        return;
    }

    cout << "Tiene vehiculo? (s/n): ";
    char tieneVehiculo;
    cin >> tieneVehiculo;
    cin.ignore(); // Ignorar el salto de línea pendiente

    if (tieneVehiculo == 's' || tieneVehiculo == 'S') {
        cout << "Ingrese placa vehicular: ";
        getline(cin, nuevoUsuario.placaVehicular);
    } else {
        nuevoUsuario.placaVehicular = "N/A";
    }

    usuarios.push_back(nuevoUsuario);
      guardarUsuariosEnArchivo(); // Guardar en el archivo cada vez que se agrega un nuevo
usuario
}

int main() {
    int opcion;
```

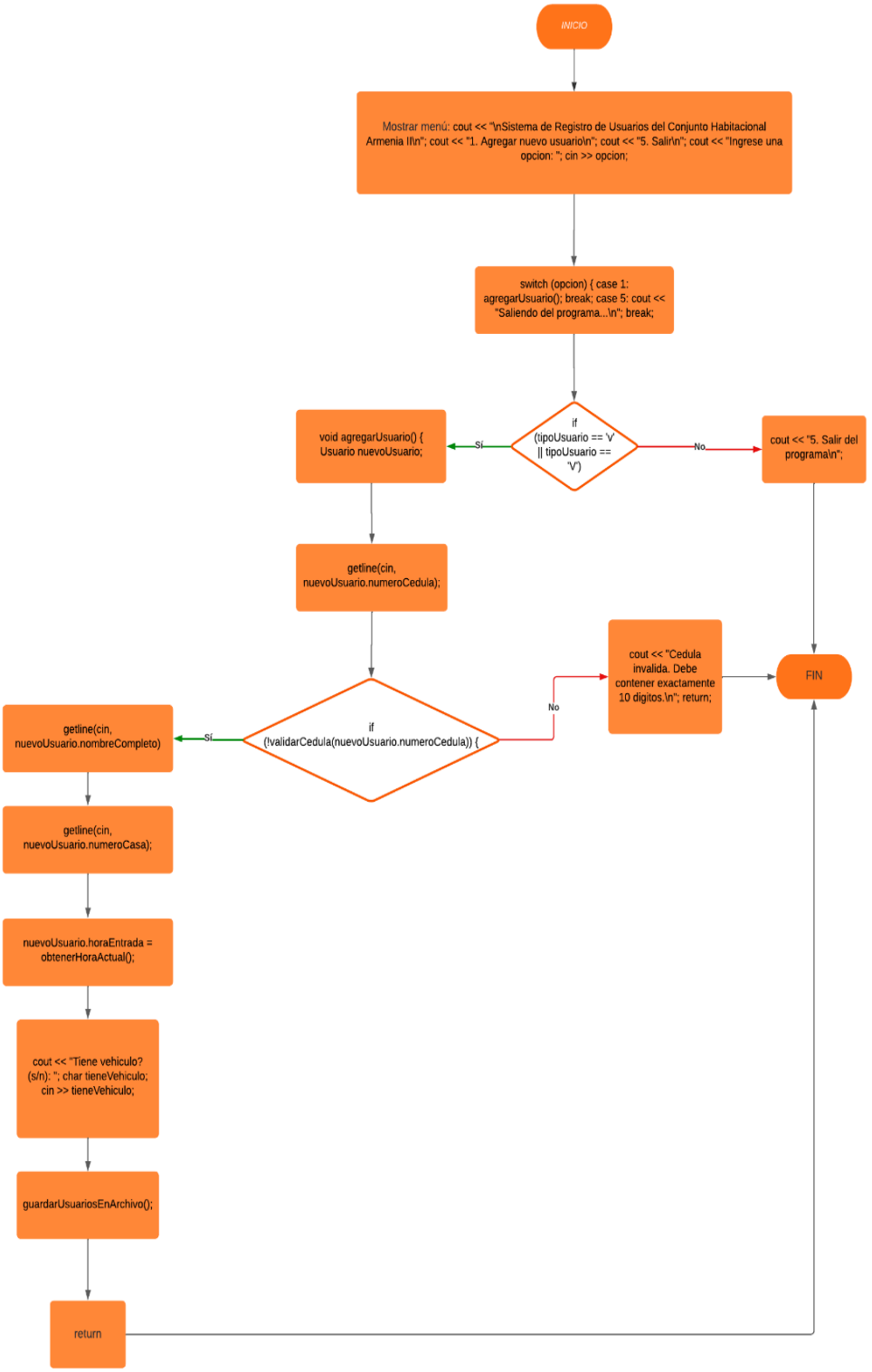
```
    do {
        cout << "\nSistema de Registro de Usuarios del Conjunto Habitacional Armenia II\n";
        cout << "1. Agregar nuevo usuario\n";
        cout << "5. Salir\n";
        cout << "Ingrese una opcion: ";
        cin >> opcion;
        cin.ignore();

        switch (opcion) {
            case 1:
                agregarUsuario();
                break;
            case 5:
                cout << "Saliendo del programa...\n";
                break;
            default:
                cout << "Opcion invalida. Intente de nuevo.\n";
        }
    } while (opcion != 5);

    return 0;
}
```

2. DIAGRAMA DE FLUJO (DF)

Realizar un DF del código fuente del numeral 1

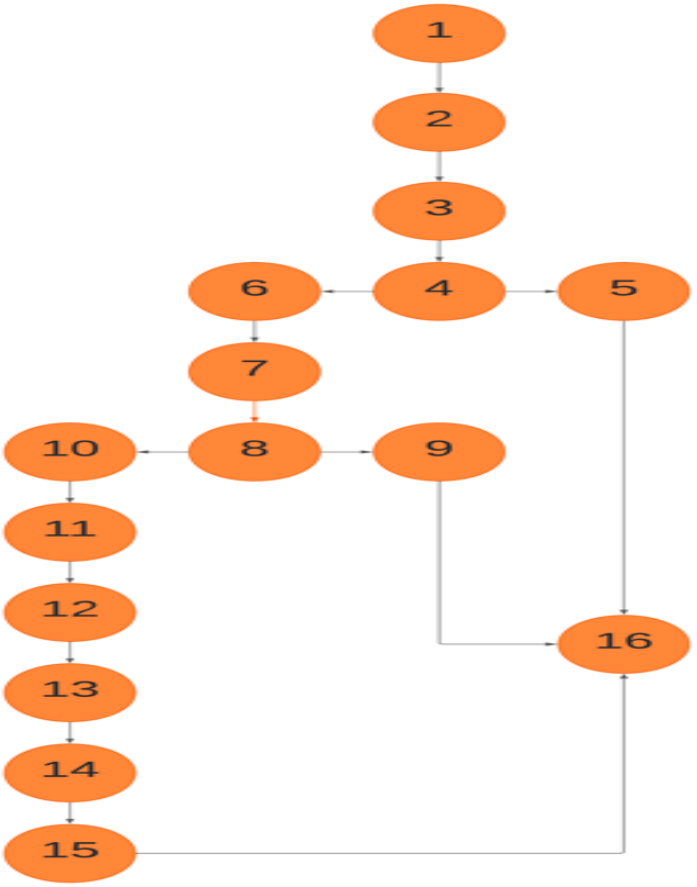
Diagrama de flujo sobre Registro de usuarios

KERLY ANDREINA CHUQUI AGUINDA | August 6, 2024



3. GRAFO DE FLUJO (GF)

Realizar un GF en base al DF del numeral 2



4. IDENTIFICACIÒN DE LAS RUTAS (Camino básico)

Determinar en base al GF del numeral 4

RUTAS

R1: 1,2,3,4,6,7,8,19,11,12,13,14,15,16
R2: 1,2,3,4,5,16
R3:1,2,3,4,6,7,8,9,16

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)= 17-16+2=3

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