**Prueba de Caja Blanca**

*“Sistema automatizado de control de Inventario”*

**Versión 1.0**

**Integrantes:**

Caizapanta Muela Tammy Amarilis

Guaiguacundo Aguirre Valeria Naomi

Pincha Llanos Estefany Anahi

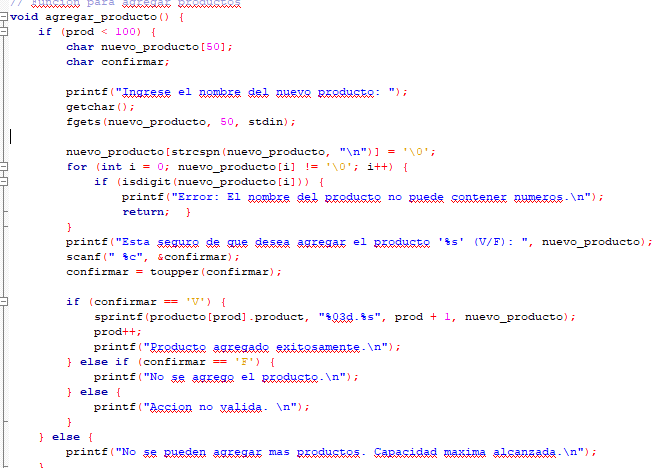
Robalino Zaldumbide Alejandro Benjamín

**Fecha**

2025-01-21

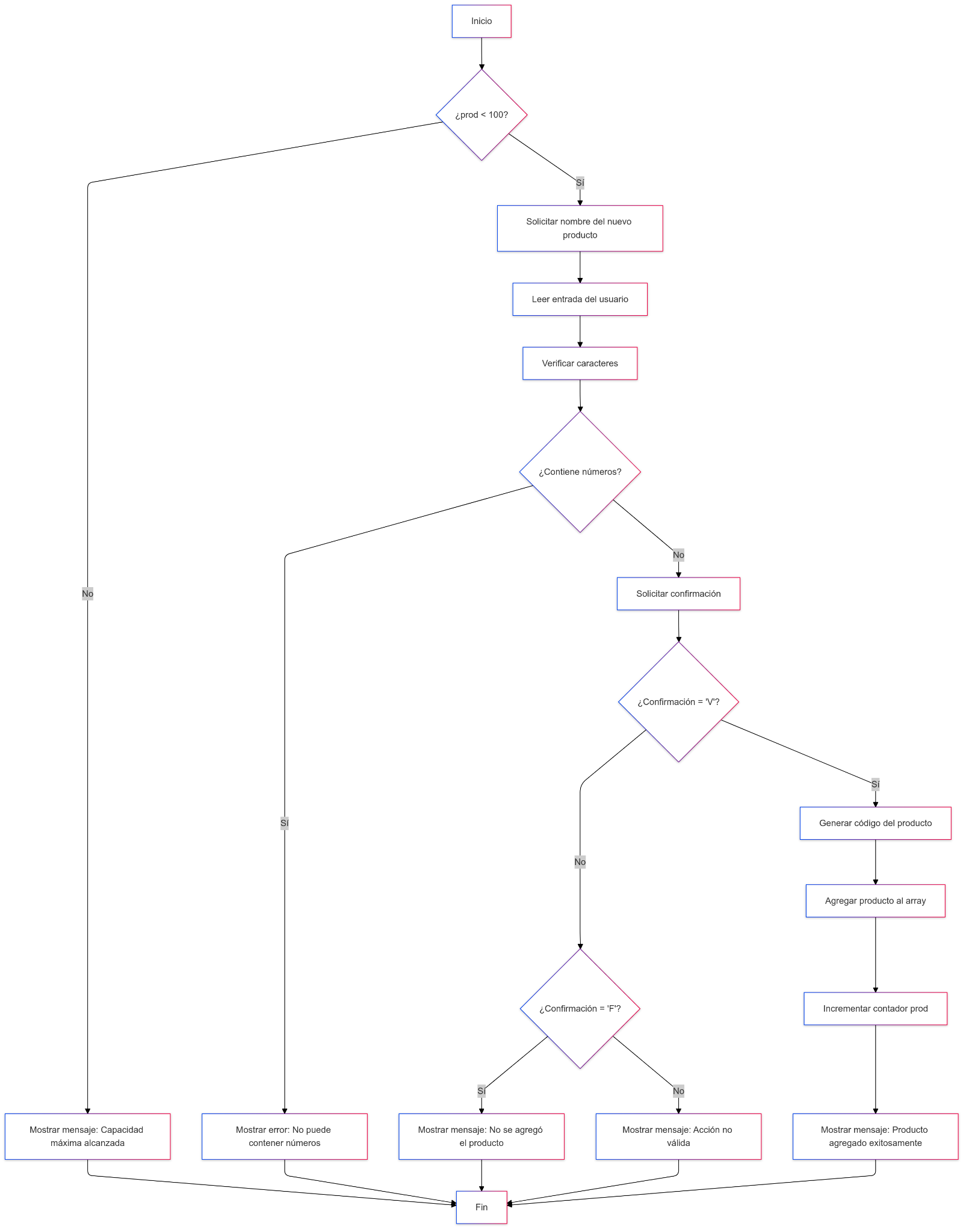
# REQ 001 - Agregar productos

## CÓDIGO FUENTE



## 

## DIAGRAMA DE FLUJO (MERMAID)



## 

## GRAFO

### 

### RUTAS

R1: 1 → 2 → 3 → 7 (confirmar == 'V')

R2: 1 → 2 → 4 → 7 (confirmar == 'F')

R3: 1 → 2 → 5 → 7 (confirmar no es ni 'V' ni 'F')

R4: 1 → 2 → 6 → 7 (capacidad máxima alcanzada)

## 

## COMPLEJIDAD CICLOMÁTICA

Por aristas y nodos:

Aristas (E) = 8

Nodos (N) = 7

V(G) = E - N + 2P

V(G) = 9 - 7 + 2(1)

V(G) = 4

Por nodos predicado:

Nodos predicado = 3

V(G) = P + 1

V(G) = 3 + 1

V(G) = 4

# REQ 002 - Buscar productos

## CÓDIGO FUENTE

// Función para buscar productos

void buscar\_producto() {

char termino[50];

int encontrado = 0;

printf("Ingrese el nombre o fragmento del producto a buscar: ");

getchar();

fgets(termino, 50, stdin);

termino[strcspn(termino, "\n")] = '\0';

printf("\nResultados de busqueda:\n");

for (int i = 0; i < prod; i++) {

if (strstr(producto[i].product, termino) != NULL) {

printf(" %d: [Producto: %s] (Cantidad: %d)\n", i + 1, producto[i].product, producto[i].cantidad);

encontrado = 1;

}

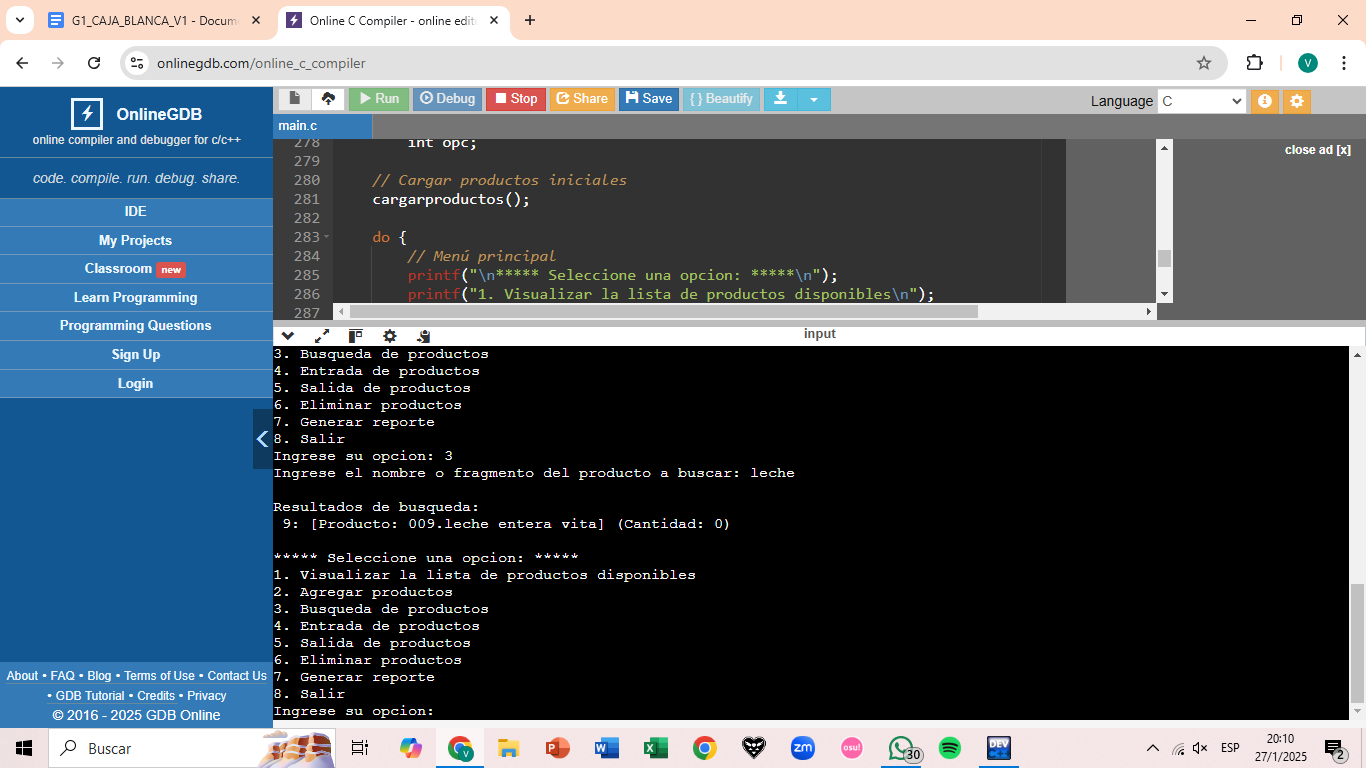
}

if (!encontrado) {

printf("No se encontraron productos que coincidan con '%s'.\n", termino);

}

}

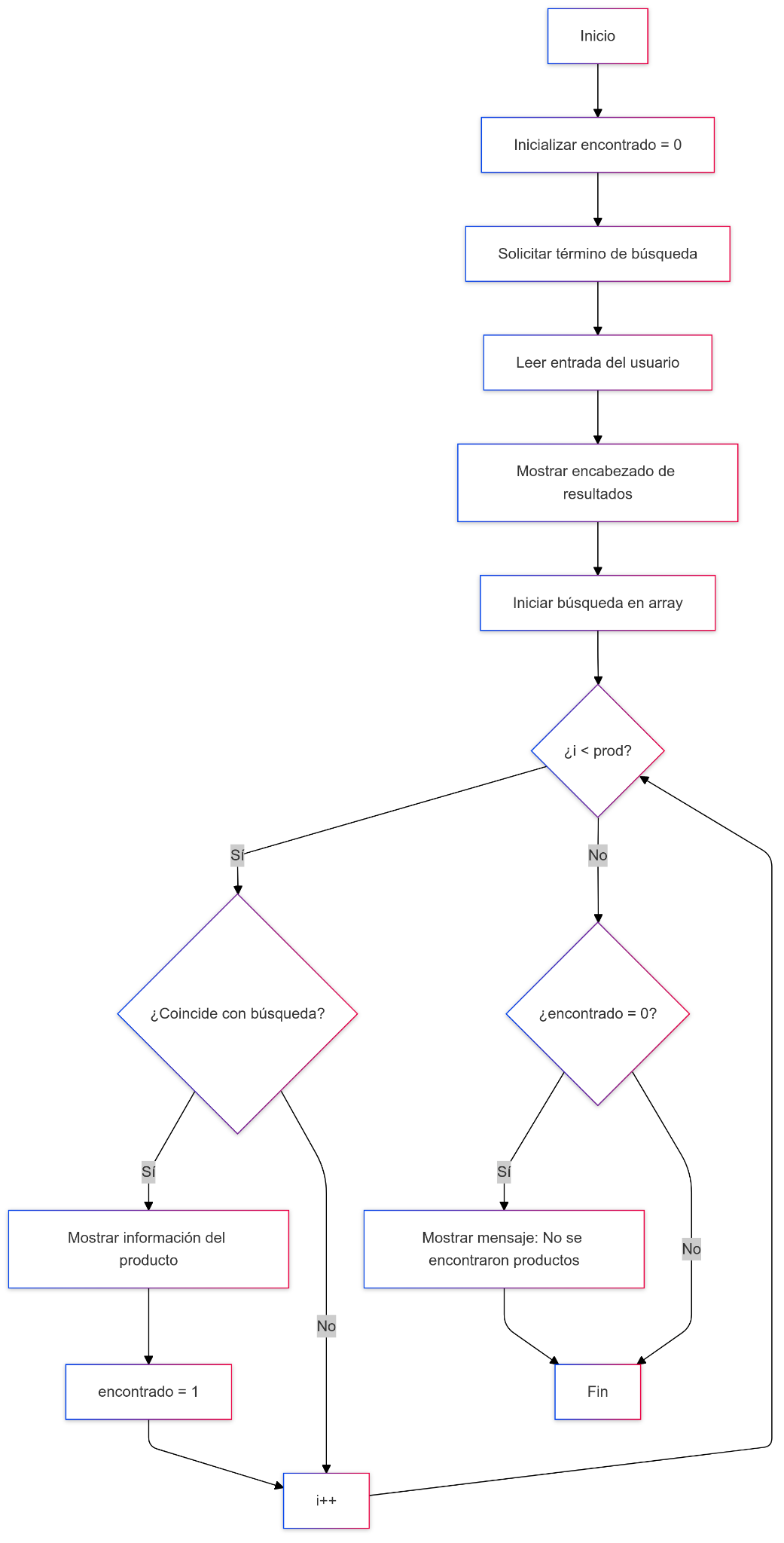


### 

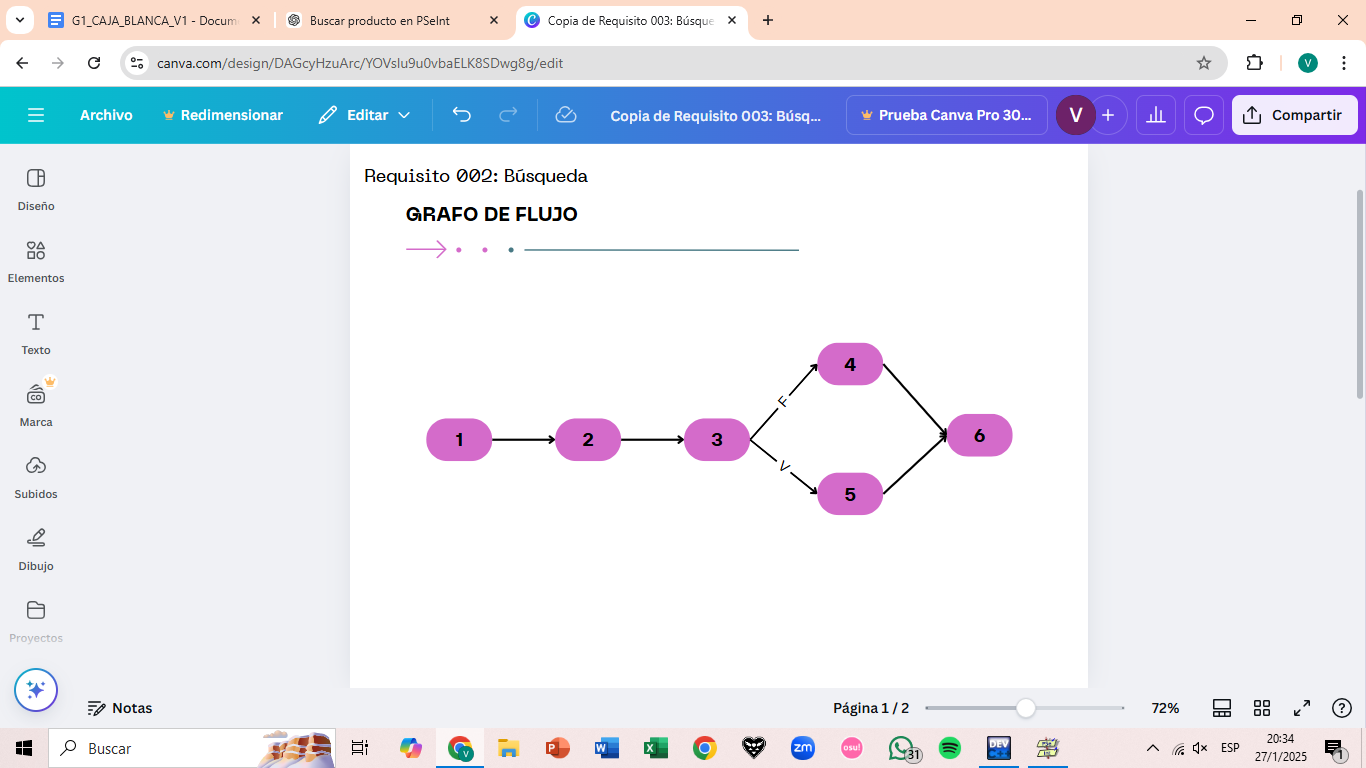
## DIAGRAMA DE FLUJO

## 

## 



## GRAFO



### RUTAS

**RUTAS:** 2

**R1:** 1–>2-->3–>4–>6

**R2:** 1–>2–>3–>5–>6

## 

## COMPLEJIDAD CICLOMÁTICA

Se puede calcular de las siguientes formas:

* V(G) = número de nodos predicados(decisiones)+1

V(G)= 1+1 = 2

* V(G) = A – N + 2

V(G)= 6 - 6 + 2 = 2

DONDE:

**P:** Número de nodos predicado

**A:** Número de aristas

**N:** Número de nodos

# REQ 003 - Entrada de productos

## CÓDIGO FUENTE

// Función para entrada de productos

void entrada\_producto() {

char termino[50];

int cantidad;

int encontrado = 0;

char confirmar;

printf("Ingrese el codigo o nombre del producto para aumentar su cantidad: ");

getchar();

fgets(termino, 50, stdin);

termino[strcspn(termino, "\n")] = '\0';

if (strlen(termino) == 0) {

printf("Error: No ingreso ningun valor. Intente nuevamente.\n");

return;

}

for (int i = 0; i < prod; i++) {

if (strstr(producto[i].product, termino) != NULL) {

printf("\nProducto encontrado: %s (Cantidad actual: %d)\n", producto[i].product, producto[i].cantidad);

printf("Ingrese la cantidad a agregar: ");

if (scanf("%d", &cantidad) != 1 || cantidad <= 0) {

printf("Error: Cantidad invalida. Solo se aceptan numeros positivos.\n");

while (getchar() != '\n');

return;

}

printf("Esta seguro de que desea agregar %d a '%s' (V/F): ", cantidad, producto[i].product);

scanf(" %c", &confirmar);

confirmar = toupper(confirmar);

if (confirmar == 'V') {

producto[i].cantidad += cantidad;

printf("\nCantidad actualizada. Nueva cantidad: %d\n", producto[i].cantidad);

} else if (confirmar == 'F') {

printf("\n No se realizo ningun cambio.\n");

} else {

printf("\nOpcion no valida..\n");

}

encontrado = 1;

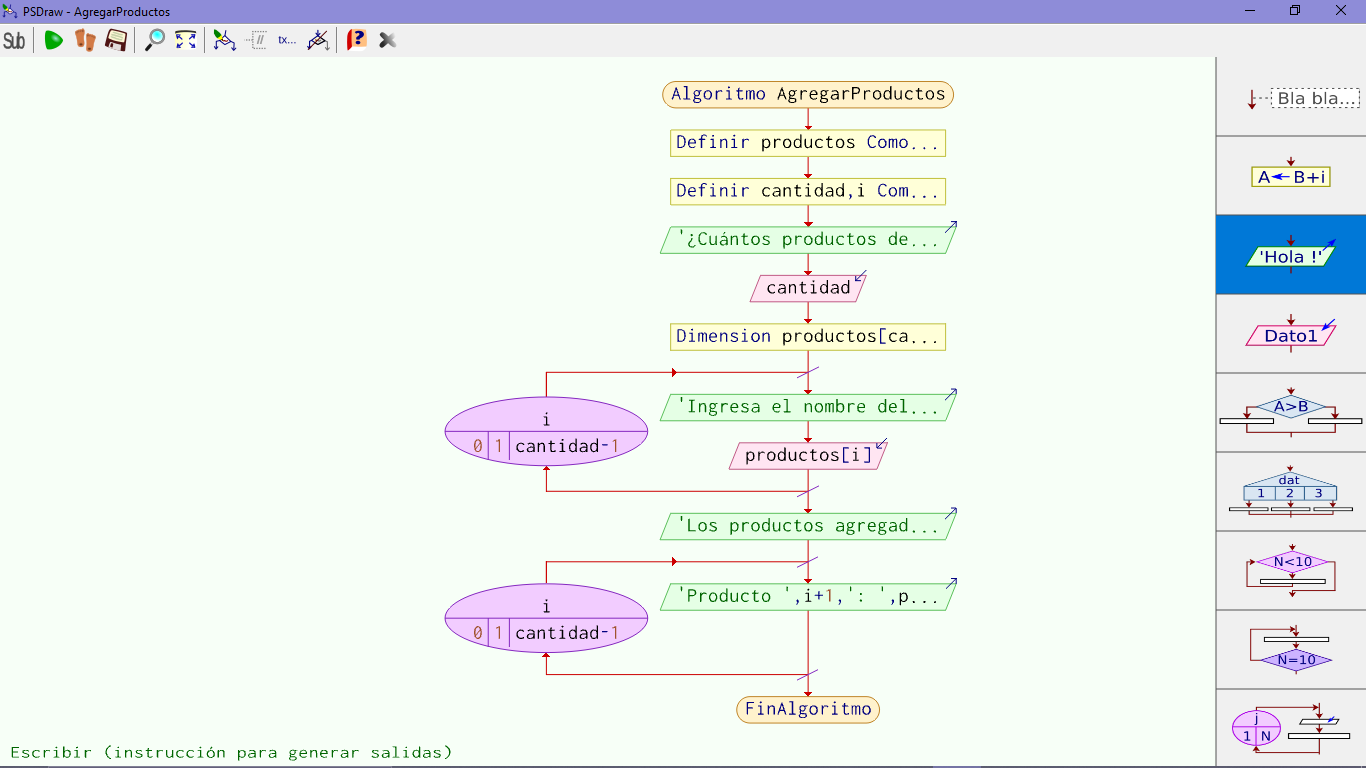
break;

}}if (!encontrado) {

printf("\nNo se encontro el producto con el codigo o nombre '%s'.\n", termino);

}

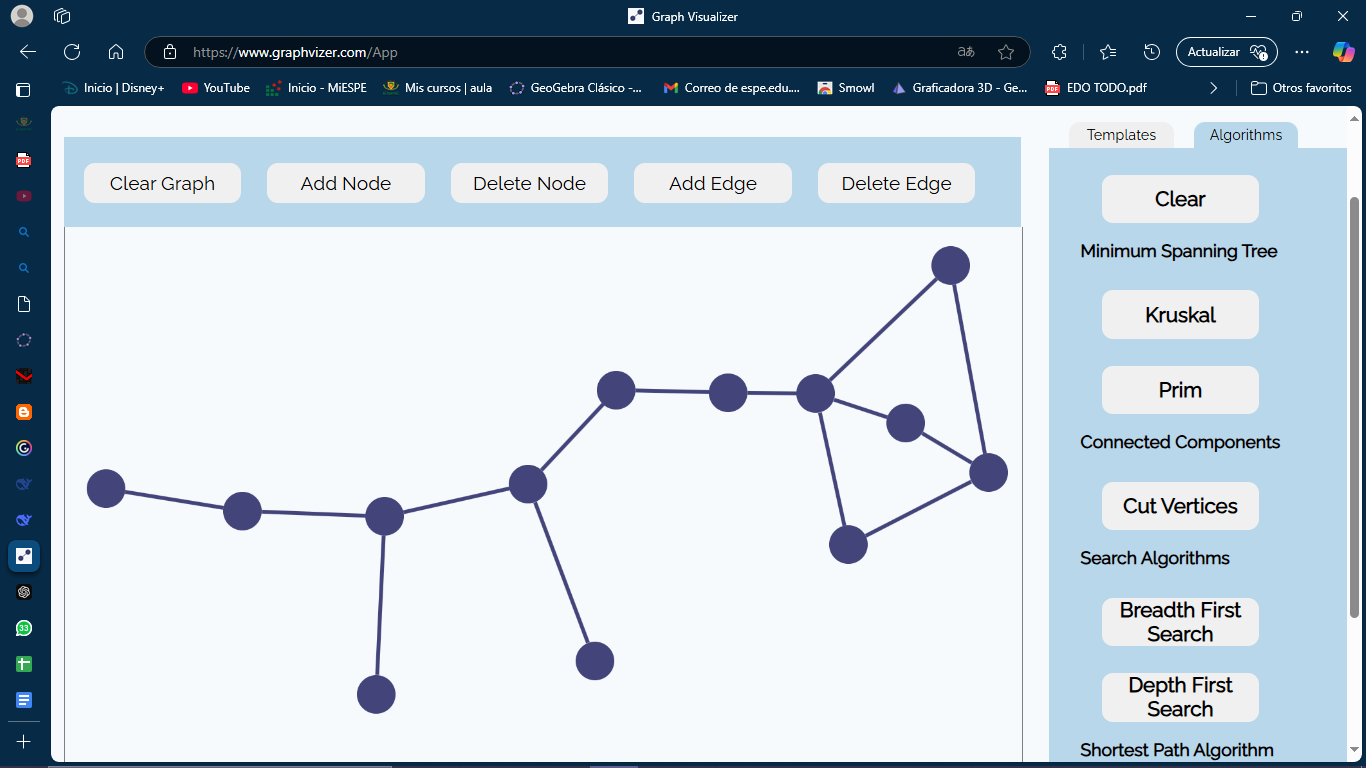
3.2 DIAGRAMA DE FLUJO



## 

## GRAFO

### 



### RUTAS

**RUTAS:7**

R1: 1 - 2 - 3 - 11

R2:1 - 2 - 3 - 4 - 5 - 10 - 11

R3:1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 11

R4:1 - 2 - 3 - 4 - 5 - 6 - 7 - 9 - 11

R5:1 - 2 - 3 - 4 - 5 - 6 - 7 - 10 - 11

R6:1 - 2 - 3 - 4 - 5 - 6 - 11

R7:1 - 2 - 3 - 4 - 5 - 10 - 1

3.3 COMPLEJIDAD CICLOMÁTICA

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

V(G)= 1+1 = 2

* V(G) = A – N + 2

V(G)= 16 - 11 + 2 = 7

DONDE:

**P:** Número de nodos predicado

**A:** Número de aristas

**N:** Número de nodos

# REQ 004 Salida de Productos

## CÓDIGO FUENTE

## 

## 

## DIAGRAMA DE FLUJO

## 

## 

## 

## GRAFO

## 

### 

### RUTAS

Ruta 1: 1 → 2 → 3 → 4 → 8

Ruta 2: 1 → 2 → 3 → 5 → 8

Ruta 3: 1 → 2 → 6 → 8

Ruta 4: 1 → 2 → 7 → 8

## COMPLEJIDAD CICLOMÁTICA

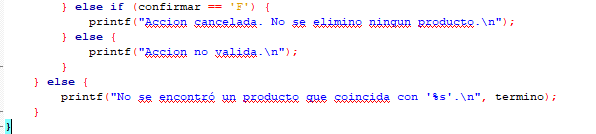
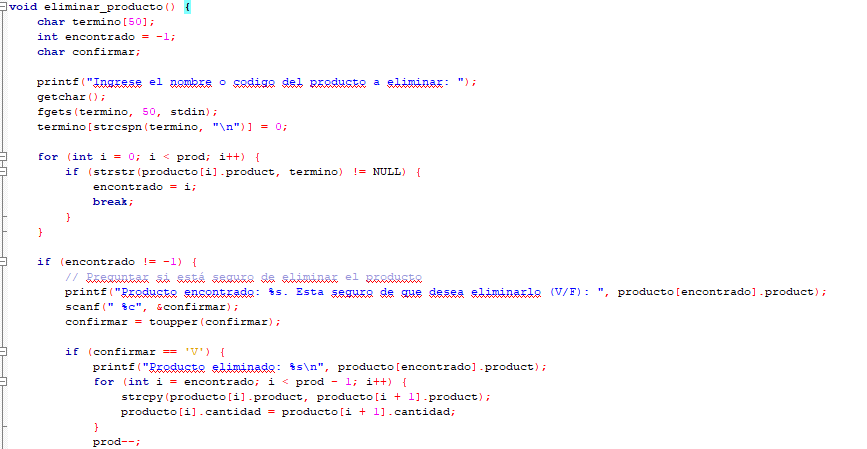
V(G) = 10 - 8 + 2(1) V(G) = 4

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

V(G)= 3+1 = 4

# REQ 005 - Eliminar Productos

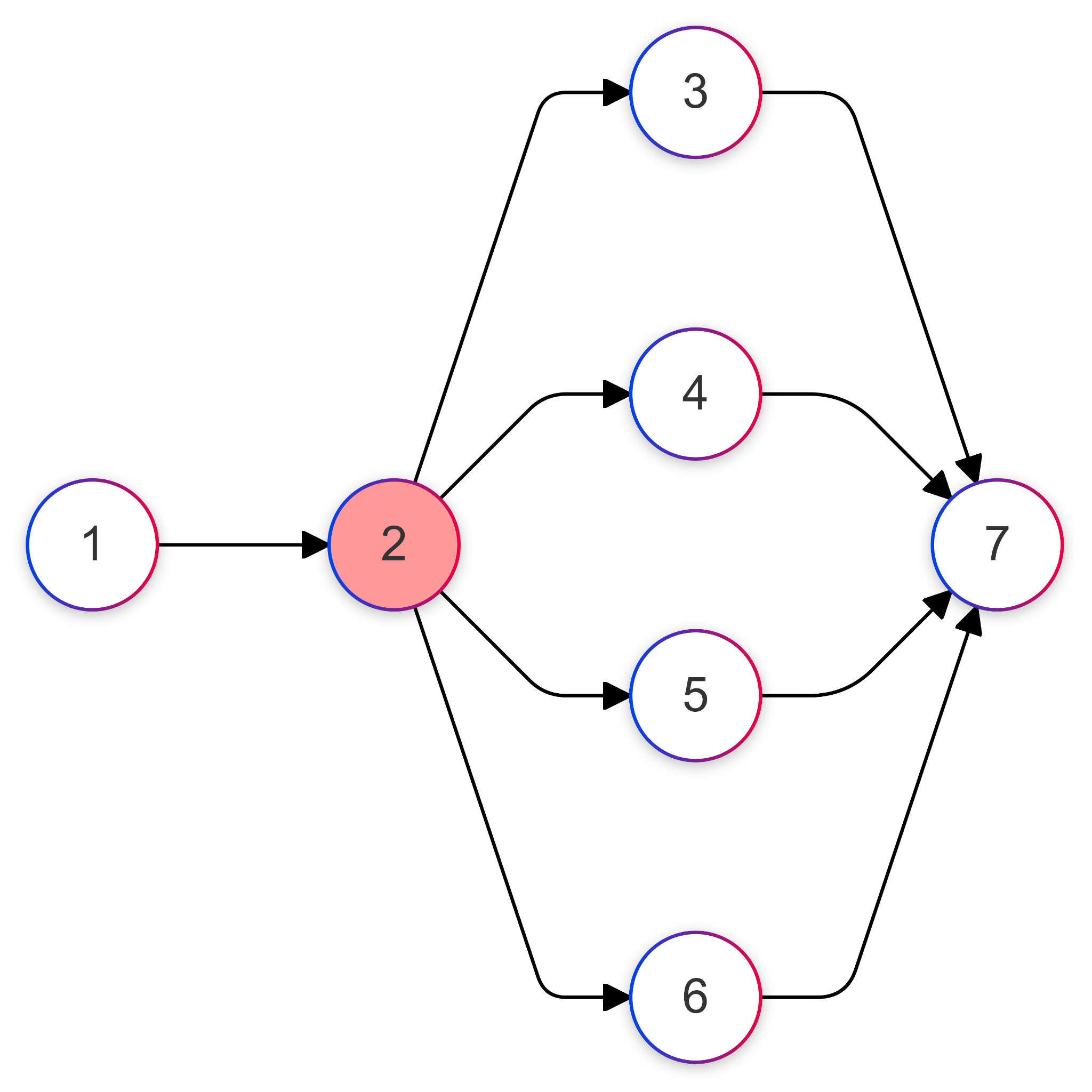
## CÓDIGO FUENTE



## DIAGRAMA DE FLUJO

## 

## GRAFO



### RUTAS

R1 : 1 → 2 → 3 → 7 (confirmar == 'V')

R2 : 1 → 2 → 4 → 7 (confirmar == 'F')

R3: 1 → 2 → 5 → 7 (confirmar no es válido)

R4 : 1 → 2 → 6 → 7 (producto no encontrado)

## 

## COMPLEJIDAD CICLOMÁTICA

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

V(G) = 3+1

V(G)=4

**V(G) = A – N + 2**

V(G) = 9-7+2=4

DONDE:

**P:** Número de nodos predicado

**A:** Número de aristas

**N:** Número de nodos

# REQ 006 Guardar

## CÓDIGO FUENTE

## //Funcion para guardar el archivo

## 

## void guardar() {

## // Abre el archivo en modo escritura

## FILE \*archivo = fopen("productos.txt", "w");

## if (archivo == NULL) {

## printf("Error al abrir el archivo.\n");

## return;

## }

## for (int i = 0; i < prod; i++) {

## fprintf(archivo, "%d: [Producto: %s]\n", i + 1, producto[i].product);

## }

## fclose(archivo);

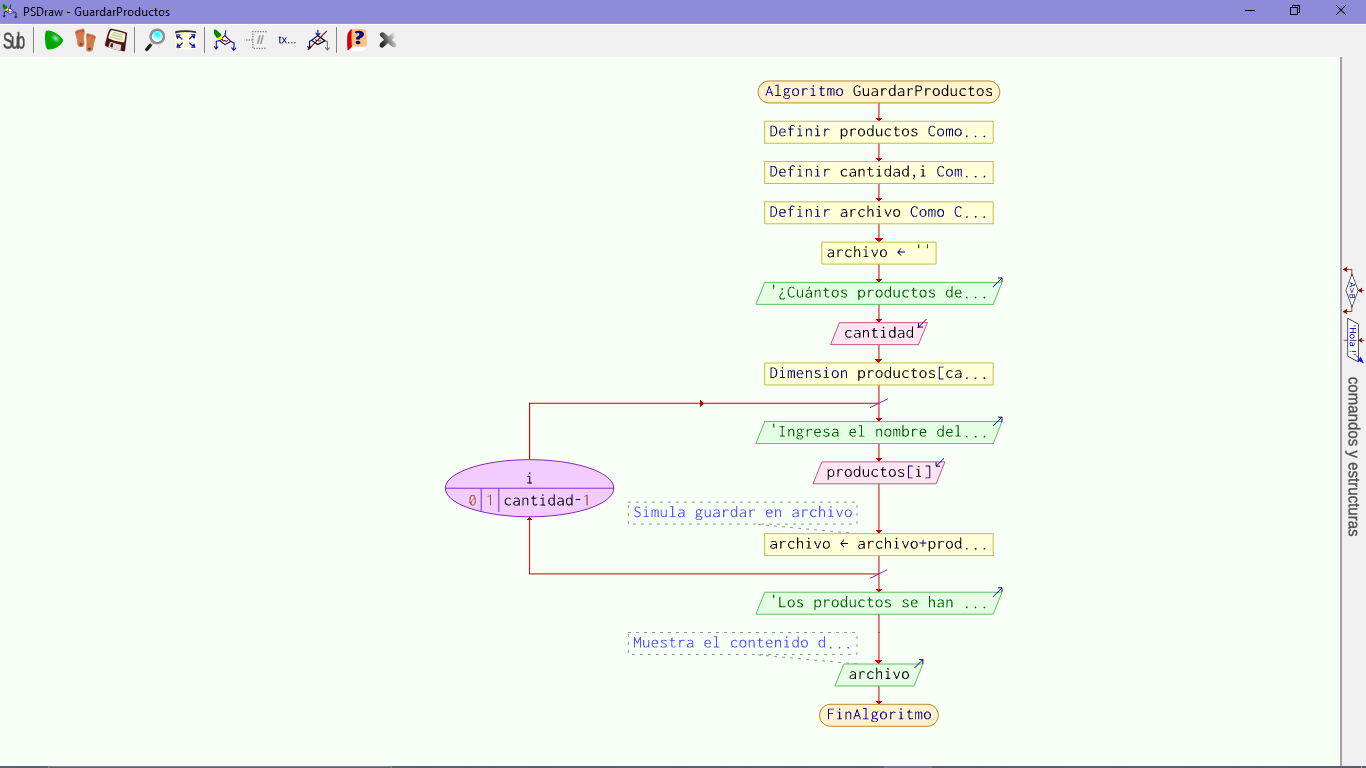
## 

## printf("Lista de productos guardada exitosamente en 'productos.txt'.\n");

## }

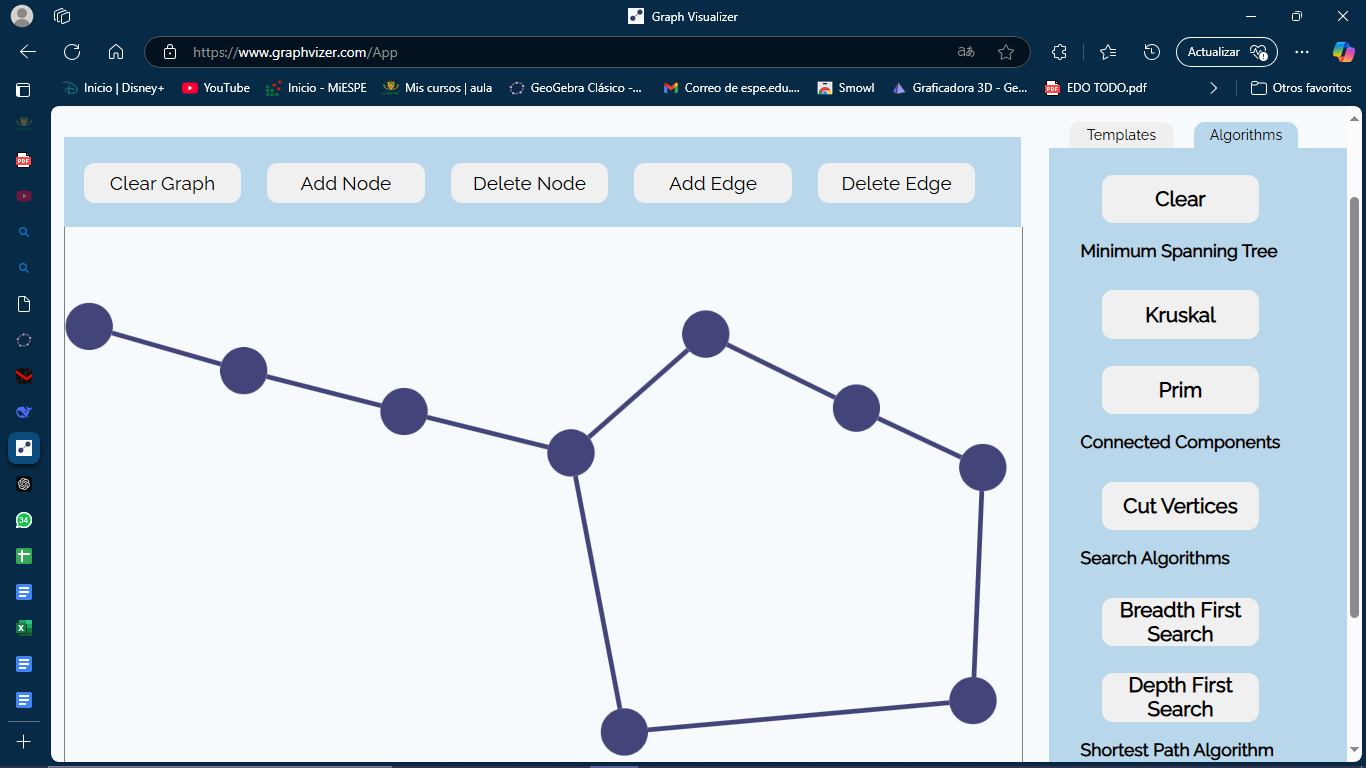
## 

## DIAGRAMA DE FLUJO



## 

## GRAFO



### 

### RUTAS

RUTAS:3

R1:1 - 2 - 3 - 8

R2:1 - 2 - 3 - 4 - 6 - 7 - 8

R3:1 - 2 - 3 - 4 - 5 - 6 - 7 - 8

## 

## COMPLEJIDAD CICLOMÁTICA

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

V(G)= 1+1 = 2

* V(G) = A – N + 2

V(G)= 9 - 8+ 2 = 3

DONDE:

**P:** Número de nodos predicado

**A:** Número de aristas

**N:** Número de nodos