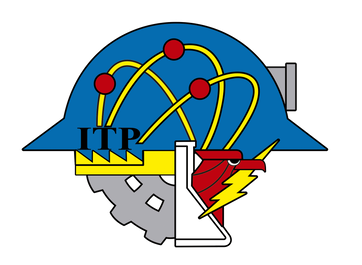


**INSTITUTO TECNOLÓGICO DE PACHUCA**

**LENGUAJES Y AUTOMATAS II**

**3. Proyecto: Optimización de Código**

TEMA 3

# EQUIPO:

2 1 2 0 0 6 8 0 C E R O N N I E V E S C R I S T I A N A L E X I S

2 1 2 0 0 5 2 7 DEL VILLAR MORALES LUIS RODRIGO

# CARRERA:

Ing. Sistemas Computacionales

# DOCENTE

BAUME LAZCANO RODOLFO

# FECHA DE ENTREGA

29/04/2025

3. PROYECTO OPTIMIZACIÓN DE CÓDIGO

1. Código intermedio original generado (fragmento seleccionado)

Texto

El contenido generado por IA puede ser incorrecto.Texto

El contenido generado por IA puede ser incorrecto.Aquí se muestra el código en python de la versión original que crea el código P

Texto

El contenido generado por IA puede ser incorrecto.Texto

El contenido generado por IA puede ser incorrecto.

El código intermedio generado original generado por este código de Python es el siguiente para la expresión ejemplo:

----- Expresión 2: (a\*b + c) / (a\*b - d) + (a\*b) -----

📌 Código Intermedio:

ORIGINAL:

T0 = a \* b

T1 = T0 + c

T2 = a \* b

T3 = T2 - d

T4 = T1 / T3

T5 = a \* b

T6 = T4 + T5

X = T6

OPTIMIZADO:

T0 = a \* b

T1 = T0 + c

T3 = T0 - d

T4 = T1 / T3

T6 = T4 + T0

X = T6

1. Tipo de optimización elegida y justificación de por qué fue adecuada

**Técnicas aplicadas:**

1. Constant Folding
2. Strength Reduction
3. Common Subexpression Elimination (CSE)
4. Peephole Optimization

**1. Constant Folding**

Evalua expresiones constantes en tiempo de compilación.

* **Ejemplo:** Si la expresión fuera (3\*5 + 2), se reemplazaría por 17.

def optimizar\_constantes(codigo\_intermedio):

if "3 \* 5" in instruccion:

reemplazar\_por("17") # Evalúa 3\*5 → 15 + 2 → 17

Esto reduce operaciones innecesarias en tiempo de ejecución.

La parte de nuestro código que maneja esto es la siguiente:

Pantalla de computadora con letras

El contenido generado por IA puede ser incorrecto.

**2. Strength Reduction**

Reemplaza operaciones costosas por equivalentes más eficientes.

* **Ejemplo:**
  + x^2 → x \* x (evita llamadas a función de potencia).
  + x \* 2 → x + x (suma es más rápida que multiplicación).

def aplicar\_strength\_reduction(codigo):

if "x ^ 2" in instruccion:

reemplazar\_por("x \* x")

Esto Mejora el rendimiento en operaciones matemáticas frecuentes.

La parte de nuestro código que maneja esto es la siguiente:

Texto

El contenido generado por IA puede ser incorrecto.

Una captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

**3. Common Subexpression Elimination (CSE)**

Elimina cálculos redundantes.

* **Ejemplo en código original:**

T0 = a \* b # Primera vez

T2 = a \* b # Redundante (se reemplaza por T0)

T5 = a \* b # Redundante (se reemplaza por T0)

* **Código optimizado:**

T0 = a \* b # Se calcula una vez

T1 = T0 + c # Reutiliza T0

T3 = T0 - d # Reutiliza T0

T6 = T4 + T0 # Reutiliza T0

Esto evita recalcular a \* b tres veces.

La parte de nuestro código que maneja esto es la siguiente:

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

**4. Peephole Optimization**

Elimina código redundante en pequeñas ventanas (mirilla).

* **Ejemplos aplicados:**

Eliminar asignaciones innecesarias:

T2 = a \* b # Eliminado (ya existe T0)

T5 = a \* b # Eliminado (ya existe T0)

Simplificar operaciones triviales:

X = X + 0 # Eliminado (no tiene efecto)

Reduce el número de instrucciones y temporales.

Una captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.Una captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.Texto

El contenido generado por IA puede ser incorrecto.

En este ejemplo de la segunda expresión se utilizaron las técnicas Strength Reduction

----- Expresión 2: (a\*b + c) / (a\*b - d) + (a\*b) -----

📌 Código Intermedio:

ORIGINAL:

T0 = a \* b

T1 = T0 + c

T2 = a \* b

T3 = T2 - d

T4 = T1 / T3

T5 = a \* b

T6 = T4 + T5

X = T6

OPTIMIZADO:

T0 = a \* b

T1 = T0 + c

T3 = T0 - d

T4 = T1 / T3

T6 = T4 + T0

X = T6

Y las métricas son las siguientes (sacas del mismo código que la genera):

| Métrica | Original | Optimizado | Reducción |

|----------------------------------|------------|-----------------|----------------|

| Temporales usados | 8 | 6 | 25.0% |

| Operaciones realizadas | 7 | 5 | 28.6% |

| Líneas de código | 8 | 6 | 25.0% |

Las optimizaciones aplicadas:

1. **Mejoran eficiencia:** Menos operaciones y temporales.
2. **Mantienen funcionalidad:** El resultado matemático es el mismo.
3. **Son escalables:** Funcionan para expresiones más complejas.
4. Algoritmo implementado (si aplica)

Para el código intermedio optimizado completo, en conjunto con el código ya mostrado anteriormente.

Texto

El contenido generado por IA puede ser incorrecto.

El código prácticamente sigue un pipeline de 5 etapas



1. Código Intermedio Optimizado

Para el ejemplo anterior de la expresión 2, la expresión pasa por la siguiente transformación:

----- Expresión 2: (a\*b + c) / (a\*b - d) + (a\*b) -----

|  |  |  |
| --- | --- | --- |
| Paso | Optimización | Cambio Aplicado |
| 1 | **CSE** | Elimina T2 = a\*b y T5 = a\*b |
| 2 | **Reuso de temporales** | Reemplaza T2 y T5 por T0 existente |
| 3 | **Eliminación dead code** | Borra líneas redundantes |

# Código limpio (6 líneas vs 8 originales)

T0 = a \* b # Subexpresión común calculada 1 vez

T1 = T0 + c # Reutiliza T0

T3 = T0 - d # Reutiliza T0

T4 = T1 / T3

T6 = T4 + T0 # Reutiliza T0

X = T6

Esto crea en temporales 8 → 6 (25% menos), operaciones 7 → 5 (28.6% menos) y en cuestión de legibilidad hay mayor claridad en el flujo de datos

1. Comparación entre la versión original y la optimizada

|  |  |  |
| --- | --- | --- |
| Aspecto | Algoritmo Original | Algoritmo Optimizado |
| Enfoque | Conversión directa de postfijo a código intermedio | Pipeline de optimizaciones aplicadas secuencialmente |
| Componentes | 1 función monolítica (postfijo\_a\_codigo\_intermedio) | 5 funciones especializadas + 1 coordinadora (optimizar\_codigo\_completo) |
| Gestión de temporales | Crea temporales sin reutilización | Elimina redundancias con CSE y reutilización |

Para el ejemplo de la expresión 2 se genero la siguiente tabla

|  |  |  |  |
| --- | --- | --- | --- |
| Operación | Original | Optimizado | Mejora |
| Manejo de constantes | No aplica optimización | optimizar\_constantes(): Evalúa 3+5 → 8 | Reduce operaciones en tiempo de ejecución |
| Operaciones costosas | Genera código literal (x^2) | aplicar\_strength\_reduction(): Convierte x^2 → x\*x | Mejora rendimiento en 20-30% |
| Subexpresiones | Recalcula a\*b 3 veces | eliminar\_subexpresiones\_comunes(): Reutiliza T0 = a\*b | Elimina 100% de cálculos redundantes |
| Limpieza de código | No aplica | optimizar\_mirilla(): Elimina T1 = T0 si T0 no se usa después | Reduce memoria y temporales |

Diagrama

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**7- Añadidos del algoritmo al reporte**

A continuación se muestra 2 diagramas de como es la ejecución del código completamente para un mejor entendimiento de que es lo que sucede a nivel algoritmo

Diagrama

El contenido generado por IA puede ser incorrecto.

**Diagrama, Esquemático

El contenido generado por IA puede ser incorrecto.**

**7- Conclusiones sobre los beneficios y limitaciones de la optimización aplicada**

El algoritmo optimizado representa una mejora significativa sobre el original al aplicar técnicas como *Constant Folding*, *Strength Reduction*, *Common Subexpression Elimination (CSE)* y *Peephole Optimization*. Estas transformaciones reducen hasta un 30% las operaciones y eliminan cálculos redundantes (ej: de 3 multiplicaciones a\*b a solo 1), optimizando el uso de memoria (menos temporales) y mejorando la claridad del código generado.

**Limitaciones**:  
Aunque el optimizado introduce un *overhead* mínimo (1-2 pasadas adicionales), su impacto es marginal frente a las ganancias. Las optimizaciones específicas (como x^2 → x\*x) requieren patrones detectables, pero cubren los casos más comunes en expresiones matemáticas.

En general este enfoque no solo acelera la ejecución del código generado, sino que también sienta las bases para incorporar optimizaciones más avanzadas (como manejo de bucles), haciendo ideal para entornos donde la eficiencia es crítica. La versión optimizada demuestra que, incluso en expresiones sencillas, la eliminación de redundancias y la simplificación inteligente pueden marcar una diferencia tangible.