INSTITUTO POLITÉCNICO NACIONAL

Unidad Profesional Interdisciplinaria de Ingeniería
Campus Zacatecas.

MATERIA:

- Ánalisis y Diseño de Algoritmos

Proyecto Final:

- Optimización del Algoritmo de Backtracking para el Problema de las N Reinas.

DOCENTE:

- Erika Sánchez Femat

Alumnos:

- Gael García Torres
- Luis Eduardo Espino Gutierrez
- Fernando Antonio Garcia Ruiz
- Javier Alejandro Martinez Torres

Introduccion

El presente proyecto se enfoca en la implementación y optimización del algoritmo de backtracking en el lenguaje de programación Python con el propósito de resolver el conocido problema de las N Reinas. Este desafío plantea encontrar la disposición óptima de N reinas en un tablero de ajedrez de NxN, de manera que ninguna reina amenace a otra. El algoritmo de backtracking se revela como una solución eficiente para abordar este tipo de problemas combinatorios.

A lo largo de este proyecto, se explorarán estrategias específicas de optimización que buscan mejorar el desempeño del algoritmo de backtracking en términos de tiempo de ejecución. Estas optimizaciones podrían abordar aspectos como la poda de ramas innecesarias, el uso eficiente de estructuras de datos, y otras tácticas que contribuyan a acelerar el proceso de búsqueda de la solución.

Además, se llevará a cabo un análisis de rendimiento comparativo entre diferentes versiones del código implementado. Este análisis se respaldará con gráficas ilustrativas que permitirán visualizar de manera clara y objetiva el tiempo de ejecución de cada versión del algoritmo. Dichas gráficas proporcionarán una visión integral de la eficiencia relativa de las estrategias de optimización implementadas

En resumen, este proyecto tiene como objetivo principal no solo ofrecer una solución al problema de las N Reinas utilizando el algoritmo de backtracking en Python, sino también explorar

y evaluar estrategias específicas de optimización para mejorar la eficiencia del algoritmo, brindando así una perspectiva completa sobre la complejidad y rendimiento de diferentes enfoques en la resolución de problemas combinatorios.

¿Como funciona La optimización del algoritmo de backtracking para el problema de las N Reinas?

La optimización del algoritmo de backtracking para el problema de las N Reinas se refiere a la aplicación de estrategias y técnicas con el objetivo de mejorar la eficiencia y rendimiento del algoritmo en la resolución de dicho problema. El problema de las N Reinas consiste en encontrar la disposición de N reinas en un tablero de ajedrez de NxN, de manera que ninguna reina amenace a otra.

Las optimizaciones se centran en reducir la cantidad de nodos explorados durante la búsqueda del espacio de soluciones, lo cual puede ayudar a disminuir el tiempo de ejecución del algoritmo. Algunas de las estrategias comunes de optimización incluyen:

Poda de Ramas (Branch and Bound): Identificar y evitar la exploración de ramas que no conducirán a una solución válida. Por ejemplo, si se coloca una reina en una posición que ya amenaza a otra, no es necesario seguir explorando esa rama.

Heurísticas Iniciales: Aplicar heurísticas inteligentes para

guiar la colocación inicial de las reinas. Estas heurísticas pueden acelerar la convergencia hacia soluciones válidas, reduciendo así la cantidad de nodos explorados.

Optimización de Estructuras de Datos: Utilizar estructuras de datos eficientes para representar el tablero y realizar verificaciones de seguridad. Esto puede incluir el uso de bitsets, matrices eficientes, o enfoques específicos para reducir el costo computacional de las operaciones.

Paralelización: Distribuir la tarea de exploración entre múltiples procesos o hilos para aprovechar la capacidad de procesamiento paralelo y acelerar la búsqueda.

Estas optimizaciones buscan hacer el algoritmo más efectivo al reducir el número de configuraciones del tablero que deben ser exploradas, mejorando así la eficiencia en la búsqueda de soluciones válidas.

Estado: Es el que refleja la condición o estado de las restricciones que enlazan las etapas. Representa la "liga" entre etapas de tal manera que cuando cada etapa se optimiza por separado la decisión resultante es automáticamente factible para el problema completo.

Codigo del Backtracking para el Problema de las N Reinas

Este código en Python implementa y compara el rendimiento de tres estrategias para resolver el problema de las N reinas, que consiste en colocar N reinas en un tablero de ajedrez de tamaño NxN sin que se amenacen entre sí.

Sin optimización: Intenta todas las posibles combinaciones sin realizar ninguna optimización.

Estrategia de poda: Utiliza la función es seguro para podar o descartar las ramas del árbol de búsqueda que no llevan a una solución válida.

Estrategia heurística: Incorpora una heurística en la primera fila del tablero para intentar mejorar la eficiencia del algoritmo.

La función es seguro verifica si es seguro colocar una reina en una posición específica del tablero sin que se amenace a otras reinas. Se realiza un chequeo en la misma columna, diagonales izquierda y derecha.

```
import time
import matplotlib.pyplot as plt

def es_seguro(tablero, fila, columna, n):
    for i in range(fila):
        if tablero[i] == columna or \
```

```
            tablero[i] - i == columna - fila or \
            tablero[i] + i == columna + fila:
                return False
    return True
```

La función colocar reinas es la función recursiva principal que intenta colocar las reinas en el tablero. Se realiza la colocación de una reina en una fila, luego se llama recursivamente para la siguiente fila. Si se llega a la última fila (fila == n), se cuenta como una solución válida y se retrocede.

```
def colocar_reinas(tablero, fila, n,
estrategia_poda=True, estrategia_heuristica=False):
    if fila == n:
        return 1

    soluciones_encontradas = 0
    for columna in range(n):
        if estrategia_heuristica and fila == 0:
            tablero[fila] = 0
        else:
            tablero[fila] = columna

        if not estrategia_poda or es_seguro(tablero,
        fila, columna, n):
            soluciones_encontradas += colocar_reinas(tab
            fila + 1, n, estrategia_poda, estrategia_heu
```

```
        tablero[fila] = -1  # Retrocede si no es posible col
        return soluciones_encontradas
```

La función resolver n reinas inicializa el tablero y llama a colocar reinas para resolver el problema. Los parámetros estrategia poda y estrategia heuristica indican si se deben utilizar las estrategias correspondientes.

```
def resolver_n_reinas(n, estrategia_poda=True,
    estrategia_heuristica=False):
    tablero = [-1] * n
    return colocar_reinas(tablero, 0, n, estrategia_poda
```

La función generar grafica utiliza la biblioteca matplotlib para generar una gráfica que muestra el tiempo de ejecución en función del tamaño del problema (N) para una estrategia dada.

La función comparar rendimiento compara las tres estrategias para diferentes tamaños de problema (N) y genera gráficas para visualizar el rendimiento de cada estrategia.

Finalmente, la función comparar rendimiento se llama para ejecutar la comparación y mostrar las gráficas.

```
def medir_tiempo(func, *args):
    inicio = time.time()
    func(*args)
    fin = time.time()
    return fin - inicio

def generar_grafica(tiempos, estrategia):
```

```
    tamanios_problema = list(range(4, 9))  # Reducir el

    plt.plot(tamanios_problema, tiempos, marker='o', lab
    plt.xlabel('Tama o del problema (N)')
    plt.ylabel('Tiempo de ejecuci n (segundos)')
    plt.title('Rendimiento del algoritmo de N reinas')
    plt.legend()
    plt.show()

def comparar_rendimiento():
    estrategias = ["Sin optimizaci n", "Estrategia de p

    for i, estrategia in enumerate(estrategias):
        tiempos_ejecucion = []
        for n in range(4, 9):  # Medir tiempos para N de
            tiempo_promedio = medir_tiempo(resolver_n_re
            tiempos_ejecucion.append(tiempo_promedio)
        generar_grafica(tiempos_ejecucion, estrategia)

comparar_rendimiento()
```

En este apartdo el codigo completo:

```
import time
import matplotlib.pyplot as plt

def es_seguro(tablero, fila, columna, n):
    for i in range(fila):
        if tablero[i] == columna or \
           tablero[i] - i == columna - fila or \
           tablero[i] + i == columna + fila:
            return False
    return True

def colocar_reinas(tablero, fila, n,
estrategia_poda=True, estrategia_heuristica=False):
    if fila == n:
        return 1

    soluciones_encontradas = 0
    for columna in range(n):
        if estrategia_heuristica and fila == 0:
            tablero[fila] = 0
        else:
            tablero[fila] = columna

        if not estrategia_poda or es_seguro(tablero,
        fila, columna, n):
            soluciones_encontradas += colocar_reinas(tab
```

```
                fila + 1, n, estrategia_poda, estrategia_heu

        tablero[fila] = -1  # Retrocede si no es posible col
        return soluciones_encontradas

def resolver_n_reinas(n, estrategia_poda=True, estrategi
        tablero = [-1] * n
        return colocar_reinas(tablero, 0, n, estrategia_poda

def medir_tiempo(func, *args):
        inicio = time.time()
        func(*args)
        fin = time.time()
        return fin - inicio

def generar_grafica(tiempos, estrategia):
        tamanios_problema = list(range(4, 9))  # Reducir el

        plt.plot(tamanios_problema, tiempos, marker='o', lab
        plt.xlabel('Tama o del problema (N)')
        plt.ylabel('Tiempo de ejecuci n (segundos)')
        plt.title('Rendimiento del algoritmo de N reinas')
        plt.legend()
        plt.show()

def comparar_rendimiento():
        estrategias = ["Sin optimizaci n", "Estrategia de p
```

```
    for i, estrategia in enumerate(estrategias):
        tiempos_ejecucion = []
        for n in range(4, 9):  # Medir tiempos para N de
            tiempo_promedio = medir_tiempo(resolver_n_re
            tiempos_ejecucion.append(tiempo_promedio)
        generar_grafica(tiempos_ejecucion, estrategia)

comparar_rendimiento()
```

Grafica sin optimizacion

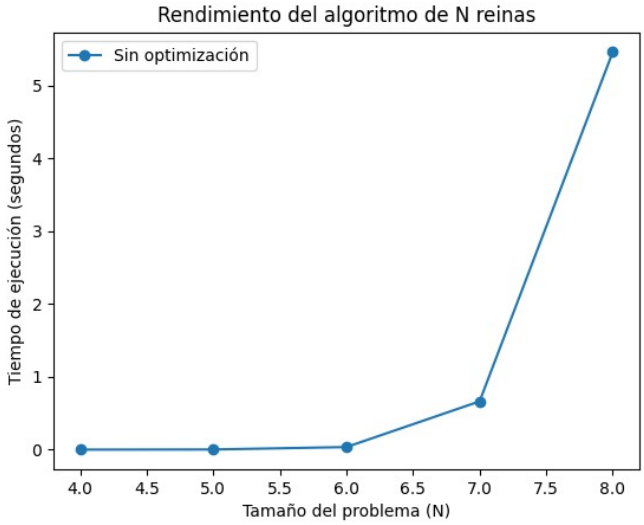

Figure 1: grafica sin optimizacion

En esta imagen podemos observar el rendimiento del algortimo de N reinas sin tener alguna optimizacion, podemos ver que en el tamaño de las reinas (de 4 a 6) el tiempo se mantiene minimo, conforme va aumentando este numero, el tiempo de ejecucion va aumentando, lo que fue de 7 a 8 reinas el tiempo se disparo rapidamente.

Grafica con estrategia de poda

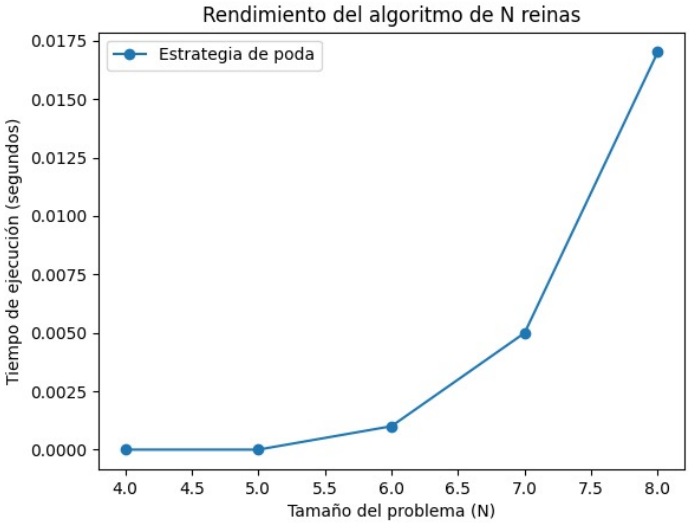

Figure 2: grafica poda

En esta imagen podemos observar el rendimiento del algortimo de N reinas con la estrategia de poda, podemos ver que en el tamaño de las reinas (de 4 a 5) el tiempo se mantiene minimo, de 5 a 6 el tiempo aumenta solo un poco, de 6 a 7 el tiempo aumento considerablemente y de 7 a 8 reinas el tiempo se disparo rapidamente.

Grafica con estrategia heuristica

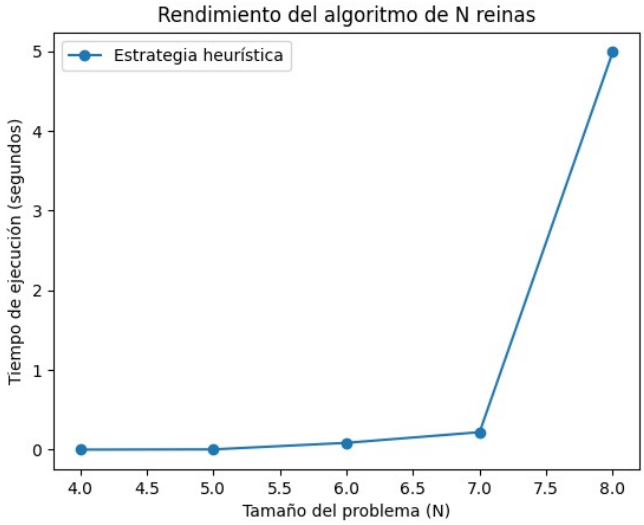

Figure 3: grafica heuristica

En esta imagen podemos observar el rendimiento del algortimo de N reinas con la estrategia heuristica, podemos ver que en el tamaño de las reinas (de 4 a 7) el tiempo se mantiene minimo, de 7 a 8 reinas el tiempo se disparo rapidamente.

Conclusiones de las graficas

Las conclusiones de las graficas pasadas es que la de estrategia de Poda era la que mas se tardara en ejecutar cuando se trataba de utilizar mas de 5 reinas, mientras que la que se comporto mejor al momneto de utilizar mas de 5 reinas fue la grafica que utilizo la estrategia heuristica, la que no tenia optimizacion se comporto entre las dos pasadas.