Implementar un algoritmo de optimización.

integrantes: Daniel Chiriboga, Daniela Tituaña, Stiven Molina, Raul Silva, Kevin Hasmal
Ejemplo práctico: Problema de las n reinas

Introducción: El problema de las n reinas plantea situaciones en las que en un tablero de ajedrez se van a situar n reinas y en ningún momento se amenacen

Objetivo General: Dar solución al problema de las reinas con un algoritmo.

Objetivos Específicos:
- Implementar algoritmos para que sitúe a las reinas en el tablero y estás no se amenacen en ningún momento
- Analizar la información recaudada en el uso del algoritmo

Marco Teórico: El problema de las n reinas consta de colocar 8 reinas en el tablero, en el cual estas no se amenacen entre sí, todo esto usando programación dinámica, y dan a conocer todos los posibles casos en los que se puedan ubicar las reinas sin que se amenacen.

Propuesta de Valor:
Se le considera como un modelo de máxima cobertura. Una solución al problema de las n-reinas garantiza que cada objeto puede ser accesado desde cualquiera de sus ocho direcciones vecinas (dos verticales, dos horizontales y cuatro diagonales) sin que tenga conflictos con otros objetos.
Algunas aplicaciones posibles son:
- Control de Tráfico Aéreo
- Sistemas de Comunicaciones
- Programación de Tareas Computacionales
- Procesamiento Paralelo Óptico
- Compresión de Datos
- Balance de Carga en un Computador multiprocesador
- Ruteo de mensajes o datos en un Computador multiprocesador

Equipo Técnico: Para este proyecto, se requiere un equipo de desarrollo compuesto por programadores con conocimientos en algoritmos y programación dinámica.

Tiempo de Desarrollo: El tiempo estimado para el desarrollo de este proyecto fue de 5 días.
Herramientas:
- Sistema de gestión de versiones: Git
- Repositorio en línea:GitHub

Explicación paso a paso:

1. **Definición del problema:** El problema se plantea con el objetivo de determinar

todas las posibles soluciones válidas para un valor dado de n. Es un problema clásico en el campo de la computación y la teoría de juegos, y se ha utilizado para explorar conceptos como la recursión, la backtracking y los algoritmos de búsqueda.

2. **Diseño del algoritmo de optimización:** la primera función “solve_n_queens(n)” toma un entero como argumento se inicializa la lista solutions en las que se van a almacenar las soluciones y también se crea el tablero, esto en forma de matriz y la llama board cada celda se representa con un punto `'.'`.
   por último llama a la función “solve_n_queens_util(n, 0, board, solutions)” esta La función nos devuelve de manera recursiva todas las posibles soluciones.

   La segunda función la cual ya invocamos en la anterior función toma como argumentos n (tamaño del tablero), col (columna actual en la que se va a colocar la reina), board (tablero actual) y soluciones (lista de soluciones encontradas). La condición para que la función termine es que col sea igual a n es decir que ya estén todas las reinas en una columna válida y esto se almacena en la lista de soluciones, el bucle repetitivo lo que hace es analizar línea por línea en el tablero. en la segunda parte de esta función hay otro bucle repetitivo, en este lo que pasa es que verifica mediante otra función “if_safe” verifica si no hay amenaza de otra reina en el tablero y marca la posición con una “Q” seguido hace una llamada recursiva de la función solo que le añade un +1 en la columna para que explore las posibilidades de la siguiente columna y por último cuando ya se hayan explorado todas las soluciones en la siguiente columna se deshacen los cambios que antes encontramos cambiando las Q nuevamente por un “.” para poder evaluar más soluciones en la misma columna.

   La tercera función “is_safe”es la encargada de dictaminar si es seguro o no colocar la reina en la celda que se analiza en la función anterior, lo primero que hace es inicializar a “n” con la longitud del tablero. el primer ciclo itera sobre cada fila de las columnas verificando si existe una “Q” retornando un False si es asi, lo mismo en el segundo ciclo solo que aqui verifica las columnas de cada fila, los siguientes dos bucles tienen como condición que mientras i sea mayor o igual que n y j mayor o igual a 0 se va a verificar de arriba hacia la izquierda en la matriz verificando que exista alguna “Q” en ese caso devuelve un False y lo mismo con el siguiente solo que las condiciones del ciclo cambian a i sea menor que n y j mayor o igual a 0 este verifica hacia abajo a la izquierda las existencias de una “Q”. para finalizar con que si no se encontró una “Q” ni en la misma fila ni en la misma columna ni en la misma diagonal se devuelve un true.

   la cuarta función toma como parámetro board que es el tablero (matriz), se inicia un bucle que itera a través del tablero se deja un espacio entre cada elemento de cada fila y se usa la función print para imprimir la cadena de cada fila en una línea nueva

   la quinta función inicializa nuevamente a n con la longitud del tablero, y se realiza un bucle anidado el primero itera a través de las filas y el segundo las columnas, dentro del segundo se verifica si contiene una “Q” y se la cambia con el carácter de una corona y si la posición no contiene una reina se imprime un “.”

3. **Implementación del algoritmo en Python**

```python
def solve_n_queens(n):
    solutions = []
    board = [['.' for _ in range(n)] for _ in range(n)]
    solve_n_queens_util(n, 0, board, solutions)
    return solutions

def solve_n_queens_util(n, col, board, solutions):
    if col == n:
        solution = []
        for row in board:
            solution.append(''.join(row))
        solutions.append(solution)
        return

    for row in range(n):
        if is_safe(board, row, col):
            board[row][col] = 'Q'
            solve_n_queens_util(n, col + 1, board, solutions)
            board[row][col] = '.'

def is_safe(board, row, col):
    n = len(board)

    # Verificar si hay una reina en la misma columna
    for i in range(n):
        if board[i][col] == 'Q':
            return False

    # Verificar si hay una reina en la misma fila
    for j in range(col):
        if board[row][j] == 'Q':
            return False

    # Verificar si hay una reina en la misma diagonal hacia
arriba
    i = row
    j = col
    while i >= 0 and j >= 0:
        if board[i][j] == 'Q':
            return False
        i -= 1
        j -= 1

    # Verificar si hay una reina en la misma diagonal hacia abajo
```

```
    i = row
    j = col
    while i < n and j >= 0:
        if board[i][j] == 'Q':
            return False
        i += 1
        j -= 1

    return True

def print_board(board):
    for row in board:
        print(' '.join(row))
    print()

def plot_solution(board):
    n = len(board)

    for row in range(n):
        for col in range(n):
            if board[row][col] == 'Q':
                print('♛', end=' ')
            else:
                print('.', end=' ')
        print()
    print()

# Ejemplo de uso
n = 8
solutions = solve_n_queens(n)

for solution in solutions:
    plot_solution(solution)
```

4. **Pruebas y depuración**: en este ejemplo se toma en cuenta un tablero de 5x5

5. **Entregables:**

- Código fuente del algoritmo implementado en Python.
- Documentación que explique el funcionamiento del algoritmo, los resultados obtenidos y cualquier mejora realizada..