



**ALUMNA:**

**BARRON ZAZUETA MARIA GUADALUPE**

**UNIDAD II**

**PROBLEMA 8 REINAS ALGORITMO TABU**

**14/03/2025**

## DESCRIPCION DEL PROBLEMA:

El problema de las 8 reinas consiste en colocar 8 reinas en un tablero de ajedrez (8x8) de manera que ninguna pueda atacar a otra. Es decir, no pueden compartir la misma fila, columna o diagonal.

## REPRESENTACIÓN DE P Y S:

- **P (Espacio de soluciones):** Conjunto de todas las configuraciones posibles de 8 reinas en el tablero.
- **S (Solución válida):** Configuración en la que ninguna reina ataca a otra.

## PROPUESTA DE ALGORITMO EN PSEUDOCÓDIGO

- Inicializar una solución aleatoria.
- Evaluar la solución calculando el número de conflictos.
- Generar vecinos mediante movimientos de reinas.
- Aplique la estrategia de búsqueda Tabú para evitar ciclos y explorar mejores soluciones.
- Continuar hasta encontrar una solución válida o alcanzar el criterio de parada.

## IMPLEMENTACION CODIGO EN PYTHON

```

1  import random
2  import time
3
4  class TabuSearch:
5      def __init__(self, n=8, max_iterations=1000, tabu_tenure=10):
6          self.n = n
7          self.max_iterations = max_iterations
8          self.tabu_tenure = tabu_tenure
9          self.tabu_list = []
10
11     def generate_initial_solution(self):
12         return [random.randint(0, self.n - 1) for _ in range(self.n)]
13
14     def calculate_conflicts(self, state):
15         conflicts = 0
16         for i in range(self.n):
17             for j in range(i + 1, self.n):
18                 if state[i] == state[j] or abs(state[i] - state[j]) == abs(i - j):
19                     conflicts += 1
20         return conflicts
21
22     def get_neighbors(self, state):
23         neighbors = []
24         for col in range(self.n):
25             for row in range(self.n):
26                 if state[col] != row:
27                     new_state = list(state)
28                     new_state[col] = row
29                     neighbors.append(new_state)

```

```

: > Users > maria > OneDrive > Documentos > import random.py > TabuSearch > tabu_search
4 class TabuSearch:
32 def tabu_search(self):
49     best_candidate = neighbor
50     best_candidate_conflicts = conflicts
51
52     if best_candidate is None:
53         break
54
55     current_state = best_candidate
56     self.tabu_list.append(current_state)
57     if len(self.tabu_list) > self.tabu_tenure:
58         self.tabu_list.pop(0)
59
60     if best_candidate_conflicts < best_conflicts:
61         best_state = best_candidate
62         best_conflicts = best_candidate_conflicts
63
64     iteration += 1
65
66     end_time = time.time()
67     return best_state, best_conflicts, iteration, end_time - start_time
68
69 if __name__ == "__main__":
70     tabu_solver = TabuSearch()
71     solution, conflicts, moves, exec_time = tabu_solver.tabu_search()
72     print(f"Mejor solución encontrada: {solution}")
73     print(f"Conflictos restantes: {conflicts}")
74     print(f"Movimientos realizados: {moves}")
75     print(f"Tiempo de ejecución: {exec_time:.4f} segundos")

```

## MÉTRICAS DE TIEMPO Y CANTIDAD DE MOVIMIENTOS:

- Medir el tiempo de ejecución del algoritmo.
- Contar la cantidad de movimientos realizados antes de llegar a la solución.