

## Programa 1: El programa de las 8 reinas

Gil Ramírez Bruno  
Moguel Silva José Miguel  
Ramirez Santamaria Isaí  
Zacarias Daniel Luis Alberto

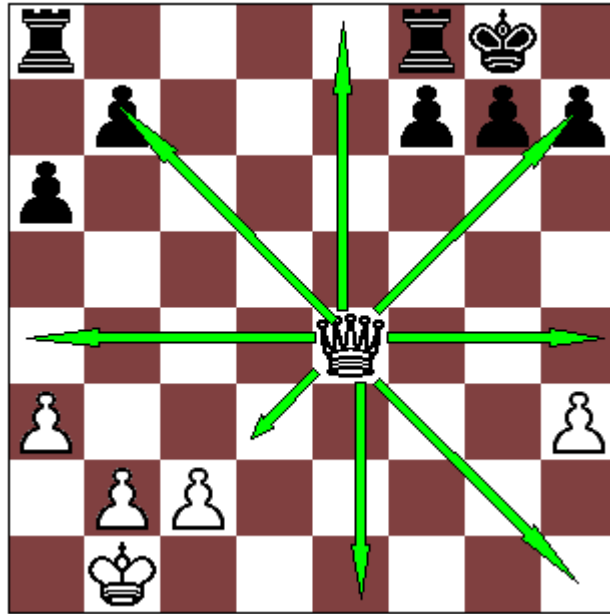
9 de Marzo de 2023

# Índice

<b>1. Planteamiento del problema</b>	<b>3</b>
<b>2. Metodología</b>	<b>3</b>
2.1. Estados inicial y final . . . . .	3
2.2. Función objetivo . . . . .	3
2.3. Función sucesor . . . . .	4
2.3.1. Orden de los operadores . . . . .	4
<b>3. Diseño y descripción del algoritmo</b>	<b>4</b>
<b>4. Reporte de la trayectoria</b>	<b>6</b>
<b>5. Análisis de complejidad</b>	<b>7</b>
<b>6. Apéndices</b>	<b>7</b>
6.1. Código del programa . . . . .	7
6.2. Manual de usuario . . . . .	10

## 1. Planteamiento del problema

El problema de las ocho reinas consiste en poner ocho reinas en el tablero de ajedrez sin que se amenacen.



Como cada reina puede amenazar a todas las reinas que estén en la misma fila y misma diagonal, cada una ha de situarse en una fila y diagonal diferente.

## 2. Metodología

### 2.1. Estados inicial y final

El estado inicial se representará en nuestro algoritmo como un vector o arreglo vacío, es decir:  $[]$ . En cuanto al estado final, este se representará como un vector o arreglo de longitud 8.

### 2.2. Función objetivo

La función objetivo solo va a buscar una cosa, que el vector o arreglo sea de tamaño 8.

### 2.3. Función sucesor

La función sucesora se encarga de generar los hijos de un estado dado. En este caso, cada hijo debe cumplir con ciertas condiciones. La primera condición es que no se puede colocar un número repetido dentro del arreglo, esto se pudiera interpretar como que no puede haber una reina en la misma columna. La segunda y tercera condición consisten en sumar y restar por cada reina existente su contenido del índice donde se encuentran y que esta no sea igual a la suma y resta del nuevo índice y potencial valor de la nueva reina. Esto se puede interpretar como que la nueva reina no se encuentra en las mismas trayectorias diagonales de las demás reinas.

#### 2.3.1. Orden de los operadores

El orden de los operadores consiste en agregar una nueva reina en la siguiente fila (iniciando en la fila 1 y terminando en la fila 8), y en cualquier columna que permita la función sucesora iniciando desde la columna 1. Es decir, cuando se inicia con un tablero vacío, o bien, un arreglo vacío en nuestro caso, existen 8 diferentes operadores (8 diferentes columnas) posibles, los cuales corresponden poner a la primera reina en cualquier columna de la primera fila. Una vez colocada, los operadores siguientes corresponden a colocar otra reina en cualquier otra columna de la siguiente fila (mientras lo permita la función sucesora) y así sucesivamente.

## 3. Diseño y descripción del algoritmo

La meta del algoritmo es encontrar una solución al problema de las 8 reinas, lo cual en nuestro caso significa que debe haber 8 reinas en el tablero sin que ninguna de ellas se amenacen. La manera en que representamos a las reinas en el tablero es a través de un arreglo con números (del 1 al 8). Cada número en el arreglo representa la columna en la que se encuentra la reina y el índice (+1) la fila. Es decir, el arreglo siguiente nos dice que hay una reina en la primera fila, segunda columna:  $\text{arreglo} = [2]$ .

Otro detalle que tomar en cuenta es que el arreglo puede ir de tamaño 0 a tamaño 8, de acuerdo con el número de reinas en el tablero, por lo que un arreglo vacío significa que no hay reinas, y un arreglo con 8 número significa que hay 8. Entendiendo lo anterior, empezamos describiendo el algoritmo, tomando en cuenta algunas cosas. Primero, la función objetivo solo va a buscar que el arreglo sea de tamaño 8, es decir que haya 8 reinas en el

tablero. Una vez que el arreglo tiene 8 reinas, sabemos que hemos llegado a una solución. Lo segundo que tomar en cuenta es la función sucesora. Esta función esta hecha de manera que sabemos que todos los hijos generados son estados que no tienen reinas que se amenazan en el juego, es decir, son estados válidos.

Ya que consideramos la función objetivo y la función sucesora, podemos diseñar la estructura general de la función que recorrerá los vectores en profundidad (Depth-First). El algoritmo se vera de la siguiente manera:

Función (tomara como argumento un vector inicialmente vacío) y retornara un booleano (True si encontró una solución o False si no):

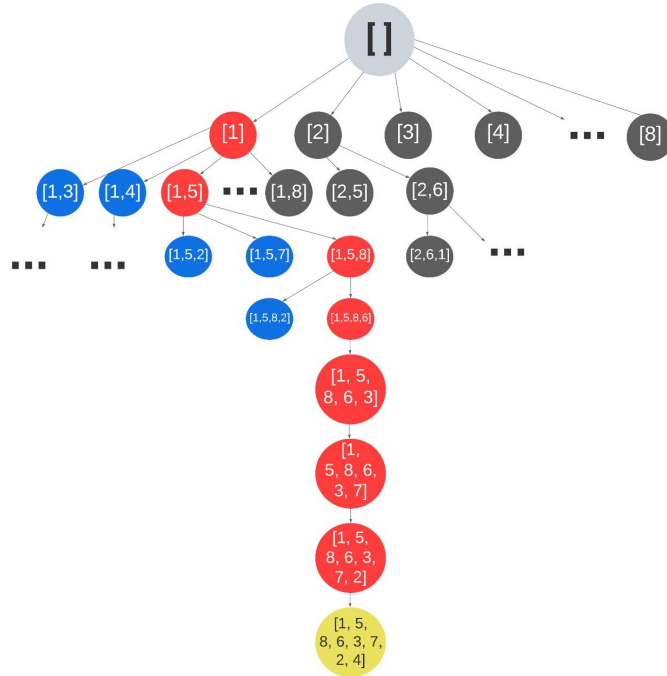
1. Ponemos al vector que se le pasa a la función en un stack
2. Entramos en un ciclo while con ciclara mientras el stack anterior no este vacío:
  - a) Hacemos pop al stack para conseguir el vector de “arriba”
  - b) Lo pasamos por la función objetivo para ver si ese es el estado objetivo
    - 1) Si sí lo es, retornamos “True”
    - 2) Si no lo es, retornamos “False”
  - c) Conseguimos todos los posibles sucesores pasándolo a la función sucesora
  - d) Agregamos todos los sucesores generados al stack creado al inicio
3. Si se sale del while, es porque se vacío todo el stack sin encontrar una solución, por lo que retornamos “False”

Una observación es que en nuestro algoritmo no utilizamos un arreglo de estados visitados. Se pudiera pensar que es necesario para no repetir estados y entrar a un árbol infinito, pero recordemos que empezamos con un arreglo vacío como estado inicial, y cada estado nuevo generado es un arreglo con una nueva reina agregada al tablero (un arreglo del mismo tamaño + 1), por lo que nunca se generara un estado ya visitado.

#### 4. Reporte de la trayectoria

La trayectoria en nuestro caso se da con la primera reina colocada en la columna 1. Por lo que la trayectoria resultante fue la siguiente:

1. []
2. [1]
3. [1, 5]
4. [1, 5, 8]
5. [1, 5, 8, 6]
6. [1, 5, 8, 6, 3]
7. [1, 5, 8, 6, 3, 7]
8. [1, 5, 8, 6, 3, 7, 2]
9. [1, 5, 8, 6, 3, 7, 2, 4]



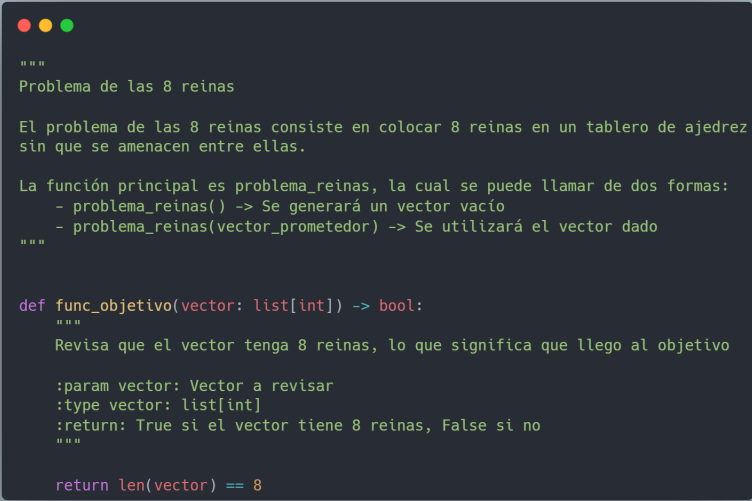
## 5. Análisis de complejidad

En cuanto a la complejidad del algoritmo, hay que tomar en cuenta algunas cosas. Primero, la arboricidad será de 8, ya que tenemos 8 posibles sucesores de un nodo. Segundo, la profundidad será 8, puesto que podemos tener máximo 8 reinas en el tablero. Entendido lo anterior, tenemos la siguiente complejidad:

$$\begin{aligned} &O(r^n y) \\ &O(r^m) \text{ donde } m = n + \log_r y \\ &n \log_r y = 8 + \log_8 1 = 8 \\ &O(r^m) = O(8^8) \end{aligned}$$

## 6. Apéndices

### 6.1. Código del programa



```
"""
Problema de las 8 reinas

El problema de las 8 reinas consiste en colocar 8 reinas en un tablero de ajedrez
sin que se amenacen entre ellas.

La función principal es problema_reinas, la cual se puede llamar de dos formas:
- problema_reinas() -> Se generará un vector vacío
- problema_reinas(vector_prometedor) -> Se utilizará el vector dado
"""

def func_objetivo(vector: list[int]) -> bool:
    """
    Revisa que el vector tenga 8 reinas, lo que significa que llego al objetivo

    :param vector: Vector a revisar
    :type vector: list[int]
    :return: True si el vector tiene 8 reinas, False si no
    """

    return len(vector) == 8
```

Figura 1: Función Objetivo

```

def func_sucesora(vector: list[int]) -> list[list[int]]:
    """
    Función que genera los sucesores del vector dado

    :param vector: Vector a revisar
    :type vector: list[int]
    :return: Lista resultante de sucesores del vector dado
    """

    hijos = []

    for i in range(8, 0, -1):
        skip = False
        copia_vector = vector.copy()

        # Asegura que no haya una reina en la misma columna
        if i in vector:
            continue

        # Asegura que no haya reina posicionada en diagonal
        for indice, v in enumerate(vector):
            if (indice + 1) - v == (len(vector) + 1) - i:
                skip = True
            if (indice + 1) + v == (len(vector) + 1) + i:
                skip = True
        if skip:
            continue

        copia_vector.append(i)
        hijos.append(copia_vector)

    return hijos

```

Figura 2: Función sucesora



```

def problema_reinas(vector_prometedor: list[int] = None) -> bool:
    """
    Función que coloca ocho reinas en el tablero de ajedrez sin que se amenacen.
    Las 8 reinas se representan con un vector, en donde el índice corresponde a
    la fila, y el número corresponde a la columna.

    Ejemplo: [1, 5, 8, 6, 3, 7, 2, 4]
    Significa que:
    - Primera fila: reina en columna 1
    - Segunda fila: reina en columna 5
    - Tercera fila: reina en columna 8
    - etc.

    :param vector_prometedor: Vector Prometedor
    :type vector_prometedor: list[int]
    :return: True si existe solución, False si no existe solución
    """
    vector_prometedor = vector_prometedor or []

    # frontera se utilizara como stack
    frontera = [vector_prometedor]

    while frontera:
        vector = frontera.pop()

        if func_objetivo(vector):
            print(vector)
            return True

        hijos = func_sucesora(vector)

        # Se agrega a la frontera los hijos del vector
        frontera.extend(hijos)

    return False

if __name__ == '__main__':
    if problema_reinas():
        print('Encontrado')
    else:
        print('No se encontró')

```

Figura 3: Función problema\_reinas y función main

## 6.2. Manual de usuario

1. Requerimientos:
  - a)* Python 3.11
  - b)* No hay necesidad de instalar librerías adicionales
2. Ejecución del programa
  - a)* Se debe estar en el directorio donde se encuentra el programa
  - b)* Abrir el cmd y ejecutar el comando `python main.py`
3. Resultados: [1, 5, 8, 6, 3, 7, 2, 4] Encontrado