8Queens - AI

June 4, 2020

1 8 Queens - Inteligencia Artificial

El problema de las Ocho Reinas (8 Queens) consiste en que en un tablero de ajedrez, 8x8 casillas, coloquemos ocho reinas de tal forma en la que ninguna se ataque entre sí. En la construcción de este algoritmo se hará una solución para un tablero de NxN y N reinas, por lo tanto se considera una solución a un problema más general.

1.1 Análisis y construcción

Para la construcción de este algoritmo primero elegimos una forma en la cual podamos representar los elementos que interactúan en él. Dada la naturaleza del problema, todas las soluciones en cada fila deben contenter una y sólo una reina, por lo tanto si encontramos todas las soluciones considerando como primer nodo cada uno de los lugares de la primera fila, al final encontraremos todas las soluciones al problema general.

Estos primer nodo serán entonces el nodo raíz en un árbol de búsqueda en el que cada nodo es la posición de cada cuadro del tablero y cada nivel del árbol de búsqueda es la fila del tablero en la que se está buscando. En el código sólo se considerará el camino actual del árbol de búsqueda.

- Estados: Cualquier combinación de cero a N reinas en el tablero en la que ninguna se ataque con otra.
- Estado inicial: El tablero vacío.
- Función sucesor: Si el tablero tiene al menos una fila vacía, se coloca una reina en la primera fila vacía tal que no tenga posibilidad de atacar a alguna otra. Si no hay lugares disponibles para poner una reina, se retira la última reina puesta y se coloca en la siguiente posición disponible de esa fila.
- Test objetivo: N reinas colocadas en el tablero.

1.2 Pseudocódigo

1. Para cada columna de fila uno, ejecuta:
    1. Sea camino = crear lista vacía
    2. Sea n = numero de columna
    3. Expandir(camino, n)
2. función Expandir(camino, n):
    1. camino.agregar(n)
    2. Si la cantidad de elementos del camino es igual al número máximo de elementos de reinas en el tablero, entonces:
        1. Guardar camino
    3. Si no, entonces:

1. Sea nodos = Generar lista de nodos alcanzables desde el estado actual
2. Para cada nodo de nodos, ejecuta:
    1. Expandir(camino, nodo)
4. camino.eliminarUltimoElemento()

1.3 Código

Para realizar el algoritmo que representa el pseudocódigo se utilizará Python 3.

Como primer paso, creamos una clase llamada Path. En esta clase se guardará el camino actual, así como los resultados obtenidos.

[1]:
```
class Path:
    def __init__(self, n):
        # Camino actual
        # Este tendrá únicamente el número de columna de la reina
        # El número de fila viene dado por la posición
        self._path = []

        # Número máximo de reinas en el tablero, por lo tanto este
        # será el límite de elementos del camino
        self._max = n

        # Conjunto de resultados
        self._result = []

        # Estas dos listas se usarán con el fin de filtrar los nodos
        # Representan las diagonales no disponibles
        self._notAvSum = []
        self._notAvRest = []

    # A continuación se presentan las propiedades de la clase Path
    # Retorna el camino
    @property
    def path(self):
        return self._path

    # Retorna el número máximo de reinas en el tablero
    @property
    def max(self):
        return self._max

    # Retorna el conjunto de resultados
    @property
    def result(self):
        return self._result

    # A continuación se presentan los métodos de la clase Path
```

```
    # Agrega un resultado al conjunto de resultados
    def addPath(self, r):
        self._result.append(r.copy())

    # Agrega un nodo al camino y actualiza restricciones
    def append(self, node):
        # Agregar restricción para columnas del tablero
        self._notAvSum.append(len(self._path) + node)
        self._notAvRest.append(len(self._path) - node)

        # Agregar nodo al camino
        self._path.append(node)

    # Elimina el último nodo agregado y actualiza restricciones
    def pop(self):
        # Quitar última restricción agregada para columnas del tablero
        self._notAvSum.pop()
        self._notAvRest.pop()

        # Eliminar último nodo del camino
        self._path.pop()

    # Retorna los nodos disponibles desde el estado actual
    def expandNode(self):
        response = []
        # i es el número de la siguiente fila
        i = len(self._path)

        # Buscar nodos
        for j in range(self._max):
            if not self._inPathOfDeath(i, j):
                response.append(j)
        return response

    # Criterios de búsqueda
    def _inPathOfDeath(self, i, j):
        aux0 = j in self._path
        aux1 = i + j in self._notAvSum
        aux2 = i - j in self._notAvRest
        return aux0 or aux1 or aux2
```

A continuación se crearán dos funciones, la primera inicializará todos los elementos que interfieren en el algoritmo e iniciará una función recursiva, la segunda es la función recursiva que buscará en el árbol de búsqueda.

[2]:
```
# n será la cantidad de reinas y dimensión del tablero
def queensDumbBattle(n):
```

```
    path = Path(n)

    # Hacemos este proceso para cada columna de la primera fila
    for i in range(n):
        expand(path, i)

    # Retornamos el resultado en formato [[(fila, columna), ..., (fila,␣
↪columna)], ...]
    return [[(i, item[i]) for i in range(len(item))] for item in path.result]
```

[3]:
```
def expand(path, root):
    # Agregar root al camino
    path.append(root)

    if len(path.path) == path.max:
        # Guardar el camino si tiene la longitud máxima
        path.addPath(path.path)
    else:
        # Obtener nodos disponibles desde el estado actual
        nodes = path.expandNode()

        # Expandir cada nodo
        for node in nodes:
            expand(path, node)
    # Eliminar root
    path.pop()
```

1.4 Resultados

Ahora vamos a ver los resultados para el tablero convencional de ajedrez:

[4]:
```
result = queensDumbBattle(8)
for item in result:
    print(item)
```

```
[(0, 0), (1, 4), (2, 7), (3, 5), (4, 2), (5, 6), (6, 1), (7, 3)]
[(0, 0), (1, 5), (2, 7), (3, 2), (4, 6), (5, 3), (6, 1), (7, 4)]
[(0, 0), (1, 6), (2, 3), (3, 5), (4, 7), (5, 1), (6, 4), (7, 2)]
[(0, 0), (1, 6), (2, 4), (3, 7), (4, 1), (5, 3), (6, 5), (7, 2)]
[(0, 1), (1, 3), (2, 5), (3, 7), (4, 2), (5, 0), (6, 6), (7, 4)]
[(0, 1), (1, 4), (2, 6), (3, 0), (4, 2), (5, 7), (6, 5), (7, 3)]
[(0, 1), (1, 4), (2, 6), (3, 3), (4, 0), (5, 7), (6, 5), (7, 2)]
[(0, 1), (1, 5), (2, 0), (3, 6), (4, 3), (5, 7), (6, 2), (7, 4)]
[(0, 1), (1, 5), (2, 7), (3, 2), (4, 0), (5, 3), (6, 6), (7, 4)]
[(0, 1), (1, 6), (2, 2), (3, 5), (4, 7), (5, 4), (6, 0), (7, 3)]
[(0, 1), (1, 6), (2, 4), (3, 7), (4, 0), (5, 3), (6, 5), (7, 2)]
[(0, 1), (1, 7), (2, 5), (3, 0), (4, 2), (5, 4), (6, 6), (7, 3)]
```

[(0, 2), (1, 0), (2, 6), (3, 4), (4, 7), (5, 1), (6, 3), (7, 5)]
[(0, 2), (1, 4), (2, 1), (3, 7), (4, 0), (5, 6), (6, 3), (7, 5)]
[(0, 2), (1, 4), (2, 1), (3, 7), (4, 5), (5, 3), (6, 6), (7, 0)]
[(0, 2), (1, 4), (2, 6), (3, 0), (4, 3), (5, 1), (6, 7), (7, 5)]
[(0, 2), (1, 4), (2, 7), (3, 3), (4, 0), (5, 6), (6, 1), (7, 5)]
[(0, 2), (1, 5), (2, 1), (3, 4), (4, 7), (5, 0), (6, 6), (7, 3)]
[(0, 2), (1, 5), (2, 1), (3, 6), (4, 0), (5, 3), (6, 7), (7, 4)]
[(0, 2), (1, 5), (2, 1), (3, 6), (4, 4), (5, 0), (6, 7), (7, 3)]
[(0, 2), (1, 5), (2, 3), (3, 0), (4, 7), (5, 4), (6, 6), (7, 1)]
[(0, 2), (1, 5), (2, 3), (3, 1), (4, 7), (5, 4), (6, 6), (7, 0)]
[(0, 2), (1, 5), (2, 7), (3, 0), (4, 3), (5, 6), (6, 4), (7, 1)]
[(0, 2), (1, 5), (2, 7), (3, 0), (4, 4), (5, 6), (6, 1), (7, 3)]
[(0, 2), (1, 5), (2, 7), (3, 1), (4, 3), (5, 0), (6, 6), (7, 4)]
[(0, 2), (1, 6), (2, 1), (3, 7), (4, 4), (5, 0), (6, 3), (7, 5)]
[(0, 2), (1, 6), (2, 1), (3, 7), (4, 5), (5, 3), (6, 0), (7, 4)]
[(0, 2), (1, 7), (2, 3), (3, 6), (4, 0), (5, 5), (6, 1), (7, 4)]
[(0, 3), (1, 0), (2, 4), (3, 7), (4, 1), (5, 6), (6, 2), (7, 5)]
[(0, 3), (1, 0), (2, 4), (3, 7), (4, 5), (5, 2), (6, 6), (7, 1)]
[(0, 3), (1, 1), (2, 4), (3, 7), (4, 5), (5, 0), (6, 2), (7, 6)]
[(0, 3), (1, 1), (2, 6), (3, 2), (4, 5), (5, 7), (6, 0), (7, 4)]
[(0, 3), (1, 1), (2, 6), (3, 2), (4, 5), (5, 7), (6, 4), (7, 0)]
[(0, 3), (1, 1), (2, 6), (3, 4), (4, 0), (5, 7), (6, 5), (7, 2)]
[(0, 3), (1, 1), (2, 7), (3, 4), (4, 6), (5, 0), (6, 2), (7, 5)]
[(0, 3), (1, 1), (2, 7), (3, 5), (4, 0), (5, 2), (6, 4), (7, 6)]
[(0, 3), (1, 5), (2, 0), (3, 4), (4, 1), (5, 7), (6, 2), (7, 6)]
[(0, 3), (1, 5), (2, 7), (3, 1), (4, 6), (5, 0), (6, 2), (7, 4)]
[(0, 3), (1, 5), (2, 7), (3, 2), (4, 0), (5, 6), (6, 4), (7, 1)]
[(0, 3), (1, 6), (2, 0), (3, 7), (4, 4), (5, 1), (6, 5), (7, 2)]
[(0, 3), (1, 6), (2, 2), (3, 7), (4, 1), (5, 4), (6, 0), (7, 5)]
[(0, 3), (1, 6), (2, 4), (3, 1), (4, 5), (5, 0), (6, 2), (7, 7)]
[(0, 3), (1, 6), (2, 4), (3, 2), (4, 0), (5, 5), (6, 7), (7, 1)]
[(0, 3), (1, 7), (2, 0), (3, 2), (4, 5), (5, 1), (6, 6), (7, 4)]
[(0, 3), (1, 7), (2, 0), (3, 4), (4, 6), (5, 1), (6, 5), (7, 2)]
[(0, 3), (1, 7), (2, 4), (3, 2), (4, 0), (5, 6), (6, 1), (7, 5)]
[(0, 4), (1, 0), (2, 3), (3, 5), (4, 7), (5, 1), (6, 6), (7, 2)]
[(0, 4), (1, 0), (2, 7), (3, 3), (4, 1), (5, 6), (6, 2), (7, 5)]
[(0, 4), (1, 0), (2, 7), (3, 5), (4, 2), (5, 6), (6, 1), (7, 3)]
[(0, 4), (1, 1), (2, 3), (3, 5), (4, 7), (5, 2), (6, 0), (7, 6)]
[(0, 4), (1, 1), (2, 3), (3, 6), (4, 2), (5, 7), (6, 5), (7, 0)]
[(0, 4), (1, 1), (2, 5), (3, 0), (4, 6), (5, 3), (6, 7), (7, 2)]
[(0, 4), (1, 1), (2, 7), (3, 0), (4, 3), (5, 6), (6, 2), (7, 5)]
[(0, 4), (1, 2), (2, 0), (3, 5), (4, 7), (5, 1), (6, 3), (7, 6)]
[(0, 4), (1, 2), (2, 0), (3, 6), (4, 1), (5, 7), (6, 5), (7, 3)]
[(0, 4), (1, 2), (2, 7), (3, 3), (4, 6), (5, 0), (6, 5), (7, 1)]
[(0, 4), (1, 6), (2, 0), (3, 2), (4, 7), (5, 5), (6, 3), (7, 1)]
[(0, 4), (1, 6), (2, 0), (3, 3), (4, 1), (5, 7), (6, 5), (7, 2)]
[(0, 4), (1, 6), (2, 1), (3, 3), (4, 7), (5, 0), (6, 2), (7, 5)]
[(0, 4), (1, 6), (2, 1), (3, 5), (4, 2), (5, 0), (6, 3), (7, 7)]

```
[(0, 4), (1, 6), (2, 1), (3, 5), (4, 2), (5, 0), (6, 7), (7, 3)]
[(0, 4), (1, 6), (2, 3), (3, 0), (4, 2), (5, 7), (6, 5), (7, 1)]
[(0, 4), (1, 7), (2, 3), (3, 0), (4, 2), (5, 5), (6, 1), (7, 6)]
[(0, 4), (1, 7), (2, 3), (3, 0), (4, 6), (5, 1), (6, 5), (7, 2)]
[(0, 5), (1, 0), (2, 4), (3, 1), (4, 7), (5, 2), (6, 6), (7, 3)]
[(0, 5), (1, 1), (2, 6), (3, 0), (4, 2), (5, 4), (6, 7), (7, 3)]
[(0, 5), (1, 1), (2, 6), (3, 0), (4, 3), (5, 7), (6, 4), (7, 2)]
[(0, 5), (1, 2), (2, 0), (3, 6), (4, 4), (5, 7), (6, 1), (7, 3)]
[(0, 5), (1, 2), (2, 0), (3, 7), (4, 3), (5, 1), (6, 6), (7, 4)]
[(0, 5), (1, 2), (2, 0), (3, 7), (4, 4), (5, 1), (6, 3), (7, 6)]
[(0, 5), (1, 2), (2, 4), (3, 6), (4, 0), (5, 3), (6, 1), (7, 7)]
[(0, 5), (1, 2), (2, 4), (3, 7), (4, 0), (5, 3), (6, 1), (7, 6)]
[(0, 5), (1, 2), (2, 6), (3, 1), (4, 3), (5, 7), (6, 0), (7, 4)]
[(0, 5), (1, 2), (2, 6), (3, 1), (4, 7), (5, 4), (6, 0), (7, 3)]
[(0, 5), (1, 2), (2, 6), (3, 3), (4, 0), (5, 7), (6, 1), (7, 4)]
[(0, 5), (1, 3), (2, 0), (3, 4), (4, 7), (5, 1), (6, 6), (7, 2)]
[(0, 5), (1, 3), (2, 1), (3, 7), (4, 4), (5, 6), (6, 0), (7, 2)]
[(0, 5), (1, 3), (2, 6), (3, 0), (4, 2), (5, 4), (6, 1), (7, 7)]
[(0, 5), (1, 3), (2, 6), (3, 0), (4, 7), (5, 1), (6, 4), (7, 2)]
[(0, 5), (1, 7), (2, 1), (3, 3), (4, 0), (5, 6), (6, 4), (7, 2)]
[(0, 6), (1, 0), (2, 2), (3, 7), (4, 5), (5, 3), (6, 1), (7, 4)]
[(0, 6), (1, 1), (2, 3), (3, 0), (4, 7), (5, 4), (6, 2), (7, 5)]
[(0, 6), (1, 1), (2, 5), (3, 2), (4, 0), (5, 3), (6, 7), (7, 4)]
[(0, 6), (1, 2), (2, 0), (3, 5), (4, 7), (5, 4), (6, 1), (7, 3)]
[(0, 6), (1, 2), (2, 7), (3, 1), (4, 4), (5, 0), (6, 5), (7, 3)]
[(0, 6), (1, 3), (2, 1), (3, 4), (4, 7), (5, 0), (6, 2), (7, 5)]
[(0, 6), (1, 3), (2, 1), (3, 7), (4, 5), (5, 0), (6, 2), (7, 4)]
[(0, 6), (1, 4), (2, 2), (3, 0), (4, 5), (5, 7), (6, 1), (7, 3)]
[(0, 7), (1, 1), (2, 3), (3, 0), (4, 6), (5, 4), (6, 2), (7, 5)]
[(0, 7), (1, 1), (2, 4), (3, 2), (4, 0), (5, 6), (6, 3), (7, 5)]
[(0, 7), (1, 2), (2, 0), (3, 5), (4, 1), (5, 4), (6, 6), (7, 3)]
[(0, 7), (1, 3), (2, 0), (3, 2), (4, 5), (5, 1), (6, 6), (7, 4)]
```

[5]:
```
len(result)
```

[5]: 92