Inteligencia Artificial: N Reinas

Carreón Guzmán Mariana
Martínez Ostoa Néstor Iván
Meza Ortega Fernando
Pedraza Martínez José Alberto

12 de noviembre del 2020

1 Introducción

El problema de las 8 reinas fue propuesto por primera vez por el alemán Max Bezzel bajo el pseudónimo de *Schachfreund*, y los publicó en 1848 en la revista ajedrecista *Berliner Schachzeitung*. Ya que la reina puede desplazarse de manera horizontal, vertical o diagonal las casillas que quiera (como se ejemplifica en la **Figura 1**), en realidad el problema se reduce a situar n reinas en el tablero de forma que no haya dos en la misma fila, columna o diagonal.

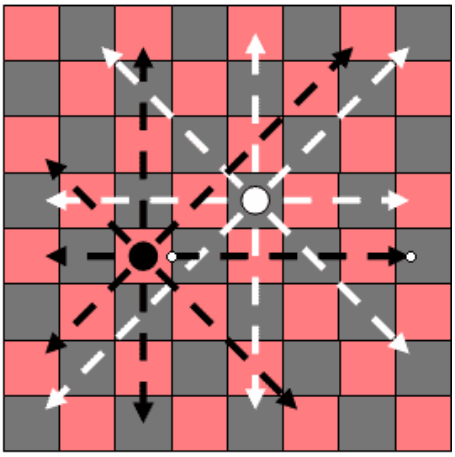

Figure 1: Posibles movimientos de la reina en el tablero.

2 Definición del problema

Como se mencionó anteriormente, el problema está basado en los movimientos que puede realizar una reina en el tablero sin que se vea afectada otra.

2.1 Descripción formal del problema

S: todos los tableros de nxn con p reinas tal que las p reinas no se ataquen y en donde $0 \leq= p \leq= n$.
$i \in S$ el estado inicial i es cualquier tablero que pertenece al conjunto S.
A: La única acción válida es poner una reina en cualquier posición del tablero tal que no exista una reina en esa posición y que no ataque a ninguna otra.
$\delta : SXA \rightarrow S$ al aplicar esta acción en un estado válido se genera otro estado válido.
$g \in S$ el estado final g es cualquier tablero en S donde el tablero tenga n reinas.
Costo de la ruta: dado que todas las soluciones tienen n reinas y poner una reina aumenta el costo en uno, todas las soluciones que se encuentren tendrán el mismo costo.

2.2 Pseudocódigo

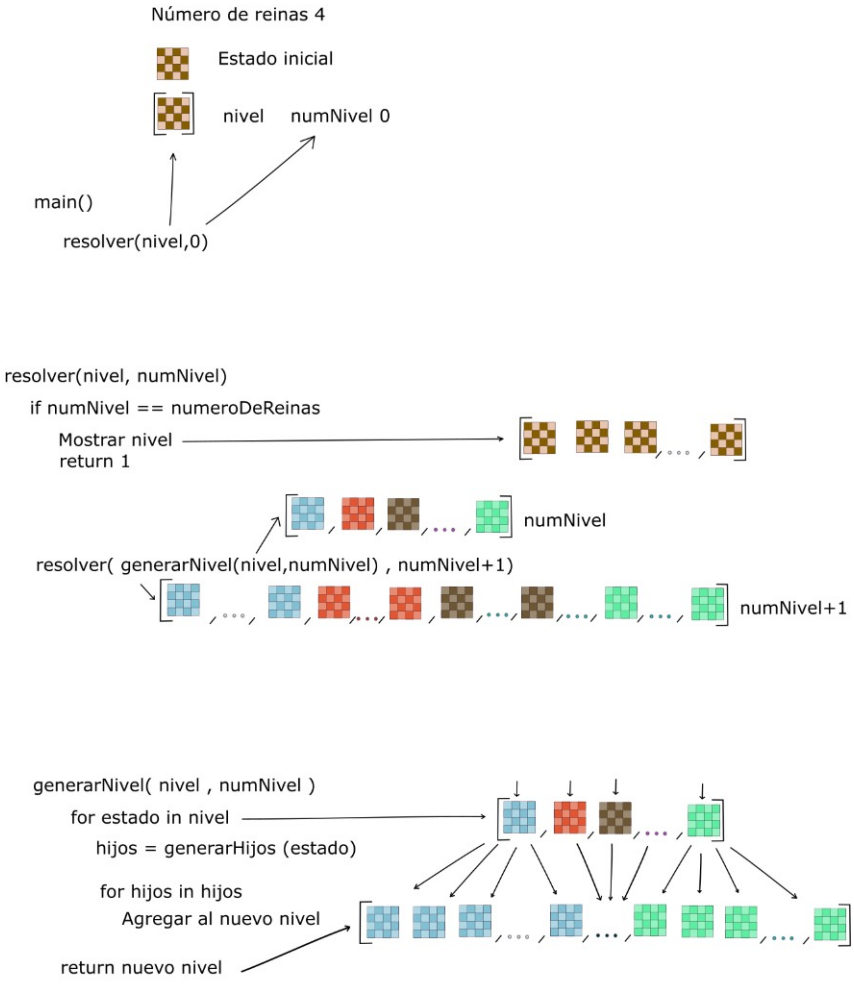

Figure 2: Definición del estado inicial, así como la forma de generar niveles.

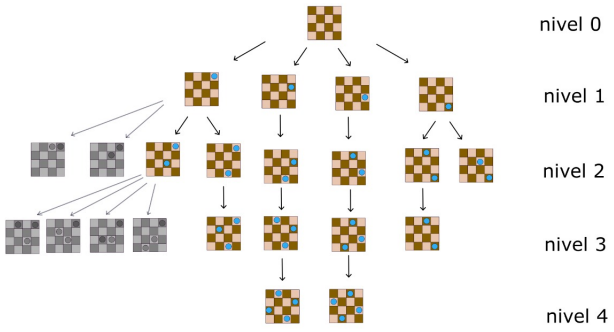

Figure 3: Definición del estado inicial, así como la forma de generar niveles.

Como podemos observar en la Figura 1 se definieron 2 funciones inicialmente con las cuales se buscaba encontrar la forma de poder resolver el problema. La primera es Resolver y con ella se definieron las condiciones a cumplir para poder decir que se había llegado a la solución del problema. En cuanto a la segunda analizamos sobre cómo se iban a ir generando los hijos, las posibles condiciones a cumplir y la forma en la que se irían almacenando.

3 Solución del problema

El problema consiste en un tablero de nxn colocar n número de reinas, de tal manera que no se ataquen y para hacerlo se usó búsqueda en anchura, lo que significa que generemos los niveles y sus hijos dependiendo del nivel anterior.
Se decidió que el estado inicial del problema fuese el tablero vacío, en este caso sería el nivel 0 e iremos generando a sus hijos hasta llegar a un nivel n. Si se llega al nivel n, entonces podemos afirmar que se ha encontrado la solución.

Generar Niveles

Para poder generar niveles es necesario examinar las columnas, se busca en las n columnas cuales son las que se encuentran vacías. En caso de estar disponible la columna, se coloca la reina y se guarda su posición. Con esto aseguramos que no haya ninguna reina en la columna para que al generar los hijos de este nivel ya no se deba verificar dicha condición.

Generar Hijos

Para generar a los hijos ya que se cumplió la condición anterior, que la columna se encuentre vacía, buscaremos que se cumplan las otras dos condiciones que son los reglones y las diagonales.Los renglones se comprobaran al hacer una comparación entre la posición donde se encuentra la reina y la nueva posición en la que se busca colocar a la siguiente reina. Aunque se genere la opción, en caso de no cumplir con la condición esta no será agregada a la lista de sus hijos. Por otra parte, para las diagonales, se deben cumplir dos condiciones ya

que se cuentan con dos diagonales. La primera es que el renglón - columna de la posición donde se encuentra la reina nos dará un valor y si queremos poner otra reina en otra posición y se realiza la misma operación renglón - columna nos deben de dar valores distintos, ya que si nos dan el mismo valor es que se encuentran en la misma diagonal. Para la otra diagonal ocurre algo similar pero ahora se deben de sumar el renglón y la columna.

4 Experimentos

Los diferentes experimentos realizados se muestran en la siguiente tabla:

4.1 BFS recursivo análisis de tiempo

n reinas	tableros encontrados	tiempo de ejecución
4	2	0.00106621
5	10	0.0038958
6	4	0.0060153
7	40	0.02097556
8	92	0.07051158
9	352	0.37461352
10	724	1.9201133
11	2680	11.578735
12	14200	68.253744
13	73712	496.771022

Table 1: Experimentos para el algoritmo recursivo

En la siguiente gráfica mostramos el tiempo de ejecución vs. el número de reinas. En estos experimentos solo probamos desde que $n = 4$ hasta $n = 13$ y podemos observar que cuando el número de reinas es 13 tenemos un tiempo de ejecución de aproximadamente 497 segundos lo que equivale 8.2 minutos.

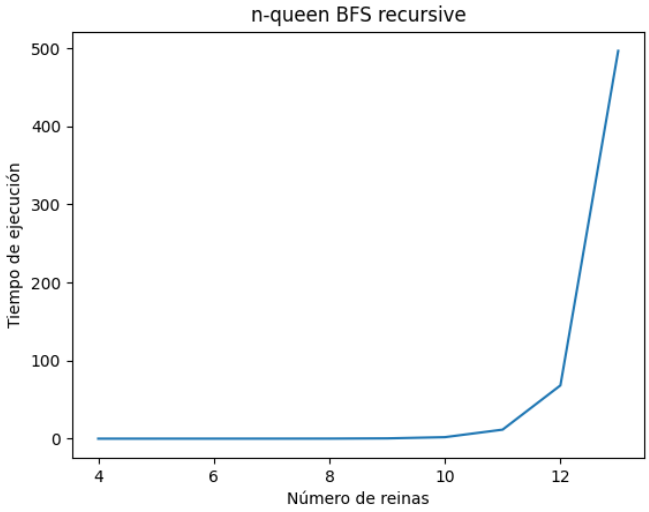

Figure 4: Número de reinas vs. Tiempo de ejecución

4.2 BFS recursivo análisis de memoria

Para hacer el análisis de tiempo lo que hicimos fue ir midiendo el cambio de memoria con respecto al tiempo; específicamente, cuando $n = 13$ comenzamos a medir desde el minuto 5 hasta el minuto 8.5 que es cuando termino de ejecutarse. Los resultados se presentan en la siguiente tabla:

Minutos	Memoria en KB
5	8281892
5.5	9186540
6	10036672
6.5	10713536
7	11340844
7.5	11912952
8	12504824
8.5	13037176
9	982236

Table 2: Minutos de ejecución vs. Memoria en KB para BFS recursivo

En la siguiente gráfica se presenta el análisis de tiempo vs memoria y podemos observar que para el minuto 9 el consumo de memoria disminuye considerablemente pues se ha llegado a una solución.

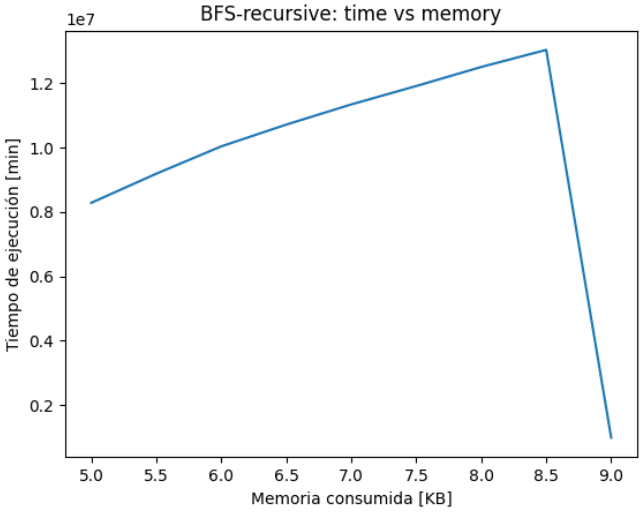

Figure 5: Tiempo vs. memoria BFS recursivo

4.3 BFS iterativo análisis de tiempo

n reinas	tableros encontrados	tiempo de ejecución
4	2	0.0010469
5	10	0.003958
6	4	0.00591302
7	40	0.0214176
8	92	0.0707808
9	352	0.3639617
10	724	1.880446
11	2680	11.4127274
12	14200	66.44955
13	73712	466.250924

Table 3: Experimentos para el algoritmo iterativo de BFS

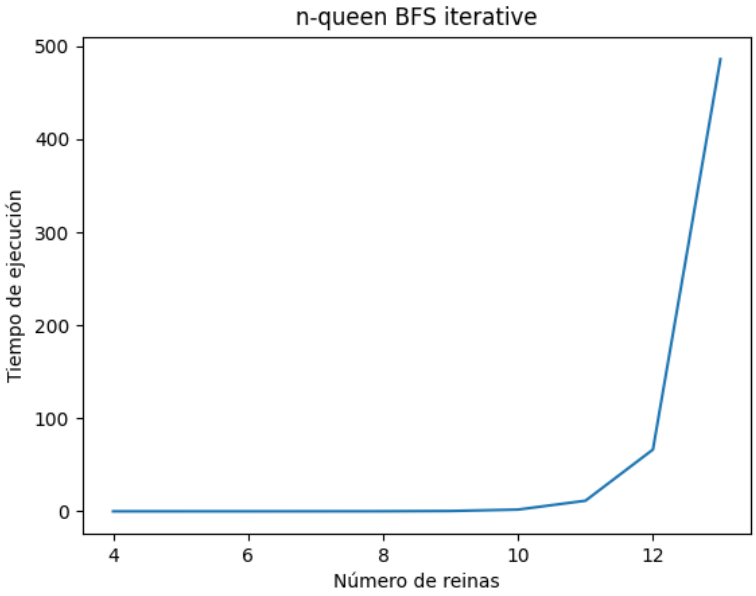

Figure 6: BFS iterativo tiempo vs número de reinas

4.4 BFS iterativo análisis de memoria

Minutos	Memoria en KB
5	5418416
5.5	6105008
6	3805140
6.5	4554196
7	5251028
7.5	2808704
8	3414808
8.5	981344
9	981344

Table 4: Minutos de ejecución vs. Memoria en KB para BFS iterativo

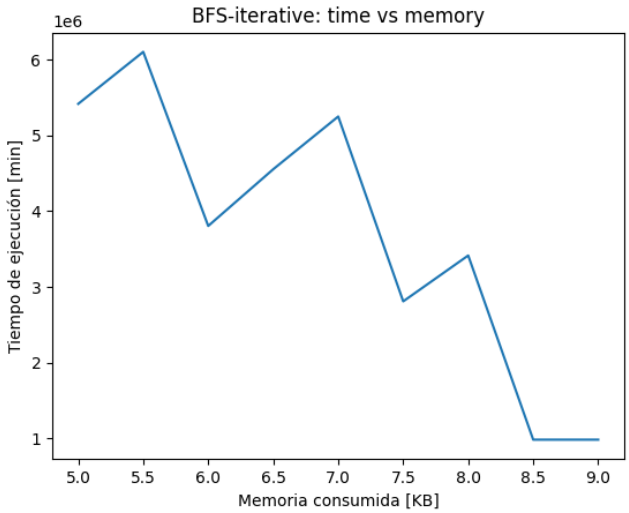

Figure 7: Tiempo vs. memoria BFS iterativo

5 Conclusiones

Es fácil observar que el BFS iterativo, aunque su desempeño en tiempo es muy similar a BFS recursivo, el consumo en memoria sí es mucho más pequeño. Para $n = 13$ después de transcurridos 8 minutos, BFS recursivo había consumido 13.03 GB mientras que BFS iterativo solo 3.41 GB.

El tiempo de ejecución en notación BigO es: $0(c^n)$ Una manera de poderlo optimizar sería si se implementara de manera iterativa, pero incluso así, por el hecho de ser exponencial sólo nos ayudaría a resolver unos cuántos casos más, después de un n no mayor a 15 es probable que el tiempo de calculo ya no sea razonable.