N Reinas

Implementado con Busqueda en Anchura

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

Universidad Nacional Autónoma de México

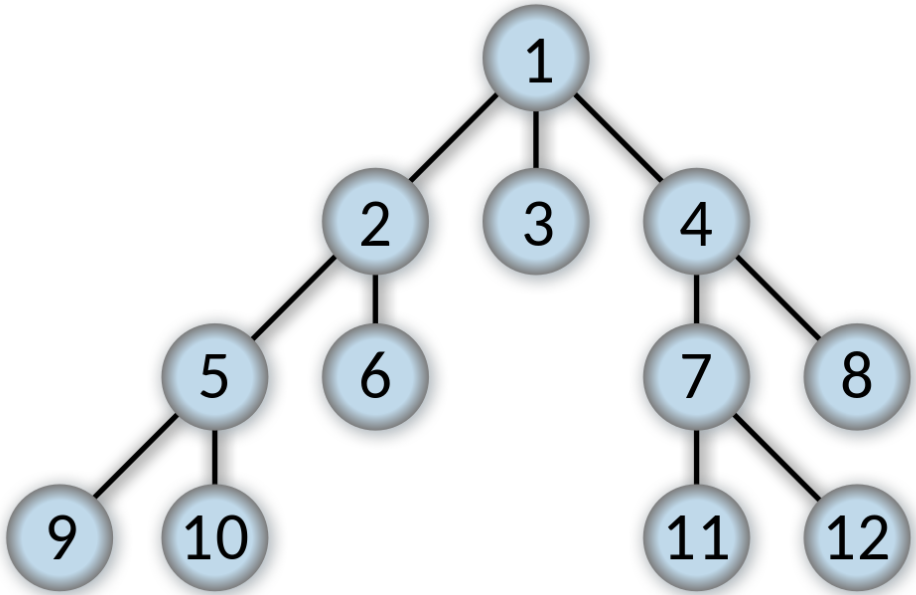
12 de noviembre de 2020

Contenido

Contenido

BFS

- La busqueda en anchura consiste en explorar el grafo comenzando en la raíz y exploranto todos los vecinos que se tengan de este, después se repite este proceso con cada uno de los vecinos del nodo raíz y así sucesivamente por cada nivel.
- Existen dos implementaciones de este algoritmo, la iterativa y la recursiva, veremos que diferencias existen entre ellas y si podemos sacar ventaja de ello.



Contenido

N Reinas

- El problema de las n reinas consiste en que se tiene un tablero de nxn, en este tablero se irán colocando una a una las reinas sin que se ataquen entre ellas hasta tener n reinas en el tablero.
- Las dos soluciones vistas en clase que existen para este problema son la busqueda en anchura y la busqueda en profundidad, veremos de manera simple cuál es la diferencia entre uno y el otro.

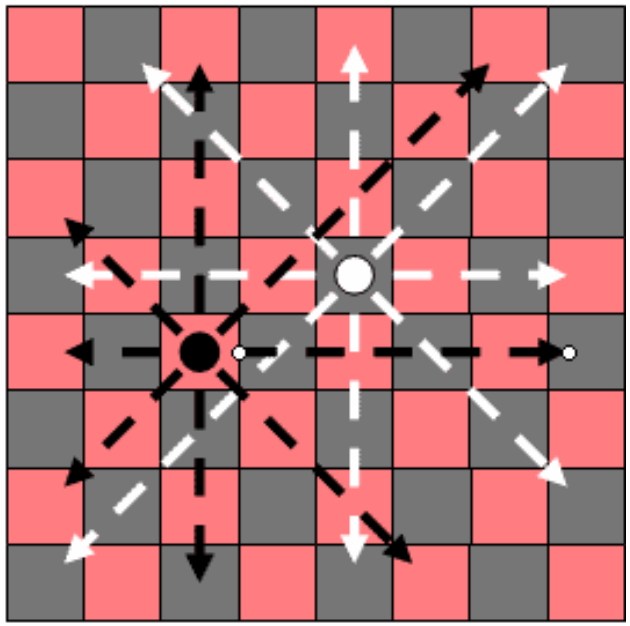


Figura: Posibles movimientos de la reina en el tablero.

Contenido

Descripción formal del problema

- S: todos los tableros de nxn con p reinas tal que las p reinas no se ataquen y en donde $1 <= p <= n$.
- $i \in S$ 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.

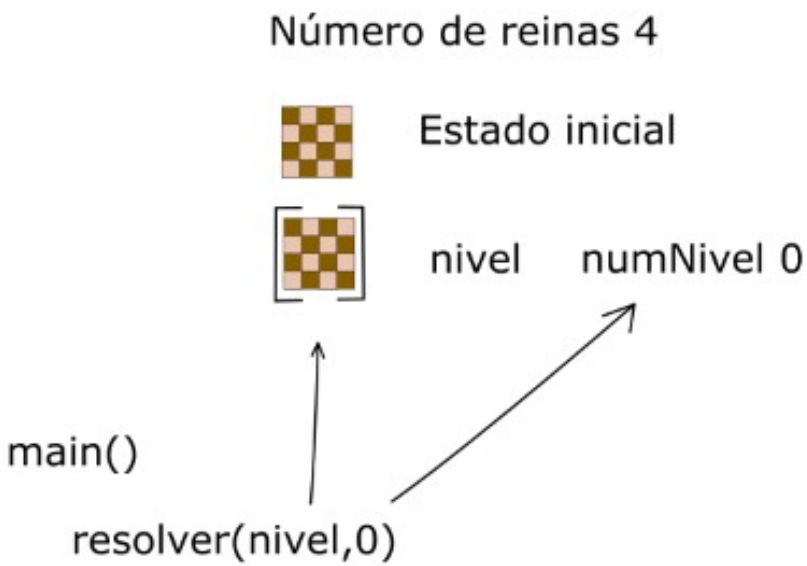
Contenido

Descripción de la solución

- La manera en que tradicionalmente se atacaría este problema con 8 reinas y aplicando busqueda en anchura sería colocando la reina en las 64 posibles posiciones, sin embargo con un análisis nos damos cuenta que esto no debe de ser así, pues al ser 8 reinas en un tablero de 8x8 nos percatamos que debe de existir una reina en cada columna y en cada fila. Esto parecería no tener relevancia pero sí la tiene.
- Al considerar que deben de existir 8 reinas en cada columna, al generar cada nivel tomaremos la primera columna vacia, el segundo nivel la segunda columna vacia y así sucesivamente; por supuesto que tendremos que checar las filas y las diagonales pero de esta forma ya no nos interesan las columnas así simplificando el algoritmo.

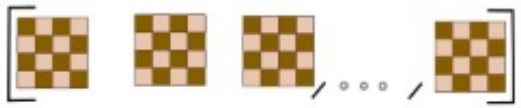
Contenido

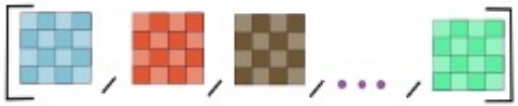
Pseudocodigo



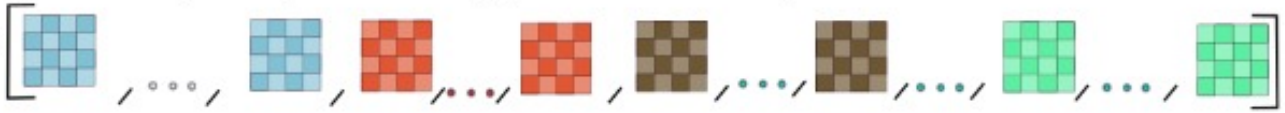
Pseudocodigo

```
resolver(nivel, numNivel)
    if numNivel == numeroDeReinas
        Mostrar nivel
        return 1
    resolver( generarNivel(nivel,numNivel) , numNivel+1)
```





numNivel



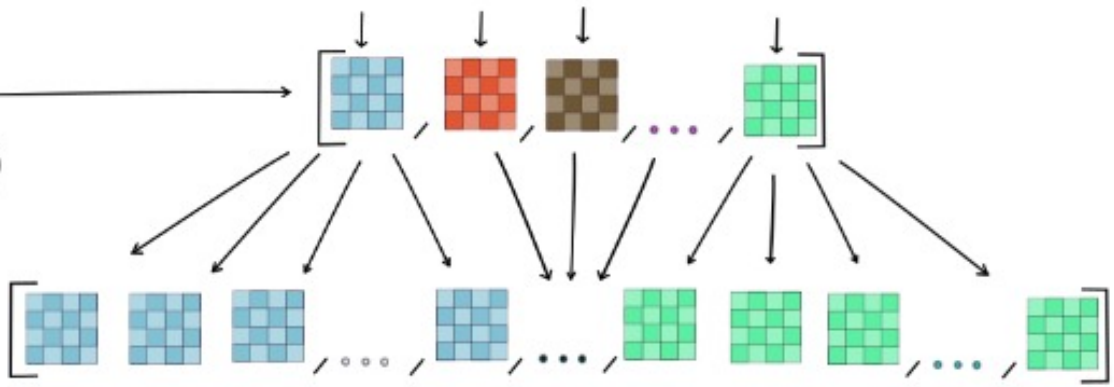
numNivel+1

Pseudocodigo

```
generarNivel( nivel , numNivel )
    for estado in nivel
        hijos = generarHijos (estado)

        for hijos in hijos
            Agregar al nuevo nivel

    return nuevo nivel
```



Contenido

Experimentos

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

Cuadro: Segundos de ejecución vs. número de reinas

Experimentos

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

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

Contenido

Complejidad algoritmica

- Deducimos que tiene una complejidad de $0(c^n)$ por los experimentos realizados.
- Este algoritmo presenta diversas desventajas, la primera de ellas es que el tiempo de ahorro entre imprimir una sola solución contra imprimir todas en notación Big O es nula, esto se debe a que el tiempo de ejecución se convierte en $0(c^{n-1} + k)$ para 1 tablero y se tarda $0(c^n)$ para todos los tableros, lo que es irrelevante en la notación Big O y ambos se comportan de manera exponencial.

Complejidad algoritmica

- Algo positivo respecto el algoritmo es que se pueden alcanzar todos los tableros relativamente rápido pero gasta incluso más memoria que DFS por tener que almacenar todo el nivel.
- DFS es mucho más eficiente para encontrar la primera solución, en contra parte para encontrar todas las soluciones es probable que sea más lento por todo el backtrack que tiene que hacer.

Preguntas