

Backtracking

Pedro O. Pérez M., PhD.

Análisis y diseño de algoritmos avanzados
Tecnológico de Monterrey

pperezm@tec.mx

08-2022

① Backtracking

- Introducción

- Algunos ejemplos

 - 8 reinas

 - Permutaciones

 - Suma de subconjuntos

 - Prime Ring Problem

- Existen problemas importantes para los cuales no se ha logrado encontrar un algoritmo eficiente para solucionarlos. Sin embargo, como son importantes, se tiene que buscar alguna forma de obtener su solución.
- En algunas ocasiones, la única herramienta con la que se cuenta es la búsqueda exhaustiva, esto es, buscar en todo el espacio de soluciones un elemento que cumpla con las condiciones para ser la solución.
- Cuando estos problemas son combinatorios, esto es, cuando su solución es una **permutación**, **combinación** o **subconjunto** de un conjunto de valores que pueden tomar las variables involucradas, el espacio de búsqueda es discreto

- La búsqueda exhaustiva podría funcionar para solucionar cualquier problema combinatorio, sin embargo, en algunos casos los espacios de solución son infinitos o muy grandes y este proceso se haría imposible de realizar en un tiempo adecuado debido a que el número posible de soluciones crece exponencialmente con el número de valores posibles de las variables.
- Backtracking es una mejora de la búsqueda exhaustiva. Va construyendo la solución paso a paso, en donde cada paso analiza los posibles valores que pueda tomar una variable. Utiliza una estructura de árbol cuya raíz es el inicio del problema y cada nivel está formado por los posibles valores que puede tomar una de las variables involucradas en el problema.

- Para no generar el árbol, utiliza el recorrido DFS.
- Al explorar un nodo, se verifica que la solución parcial hasta ese momento cumpla con las condiciones de una posible solución, si es así, este nodo es clasificado como “prometedor”, por lo que se expande y se continúa el proceso de “primero en profundidad”. Si, por el contrario, se encuentra que la solución parcial obtenida hasta ese momento no cumple con las condiciones para ser una solución al problema, ese nodo se clasifica como “no promisorio”, por lo que se desecha, se regresa a su padre.

Procedure 1 BACKTRACKING

Input: *node*

if SOLUTION(*node*) **then**

print *node*

end if

if NOT(PROMISSORY(*node*)) **then**

return

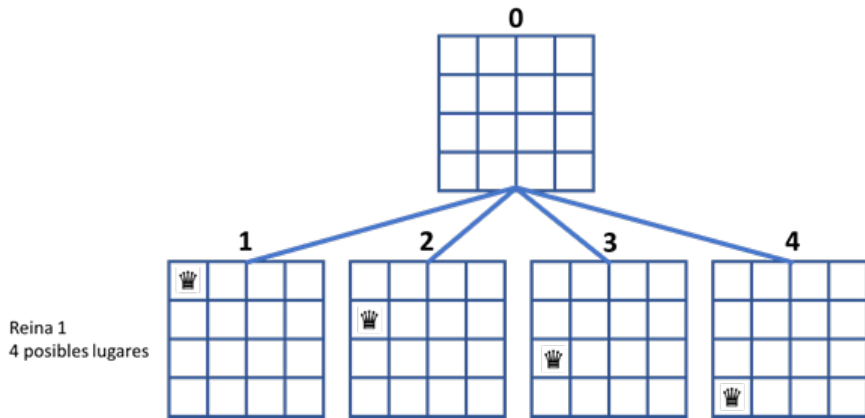
end if

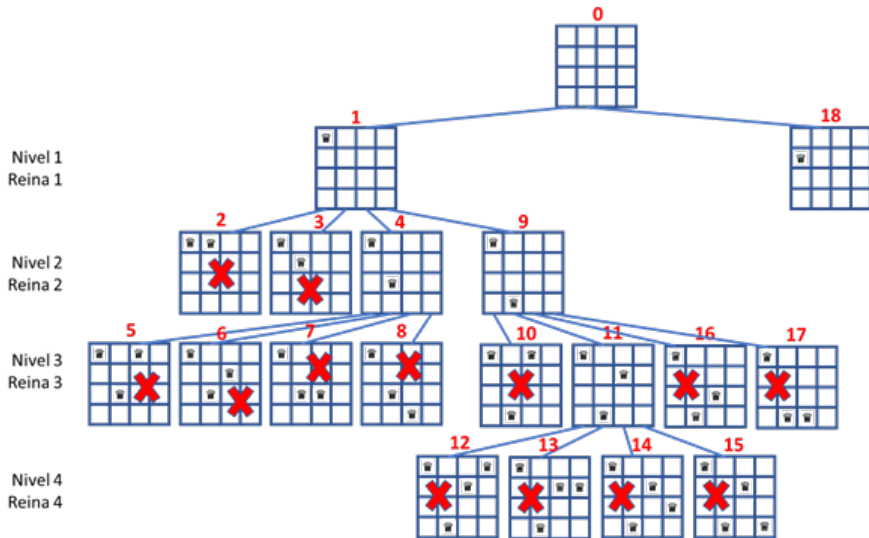
for $i \leftarrow 1$ **to** CHILDS(*node*) **do**

 BACKTRACKING(*i*)

end for

En un tablero de ajedrez (8×8 casillas) se deben colocar 8 reinas sin que se ataquen. Recordar que una reina puede atacar vertical, horizontal o diagonalmente, recorriendo cuantas casillas requiera. El problema se puede generalizar a N reinas, es decir, en un tablero de $N \times N$ casillas se colocan n reinas sin que se ataquen





Procedure 2 CAN_PLACE

Input: *rows* : Array, *col* : Integer, *ren* : Integer

for $i \leftarrow 1$ to *col* do

 if $rows[i] = ren$ or $abs(rows[i] - ren) = abs(i - c)$ then

 return *false*

 end if

end for

return *true*

Procedure 3 EIGHT_QUEENS

Input: *rows* : Array, *a* : Integer, *b* : Integer, *col* : Integer

 if $c = 8$ and $rows[b] = a$ then

 print *rows*

 end if

 for $ren \leftarrow 1$ to 8 do

 if CAN_PLACE(*rows*, *ren*, *col*) then

$rows[col] \leftarrow ren$

 EIGHT_QUEENS(*rows*, *a*, *b*, $col + 1$)

 end if

 end for

Hallar todas las permutaciones de un número N . Por ejemplo, las permutaciones de 123 son: 123, 231, 321, 312, 132, 213, 123.

Procedure 4 PERMUTATION

Input: S : *String*, pos : *Integer*

if $pos = 0$ then

print S

else

for $i \leftarrow 1$ to pos do

SWAP(S, i, pos)

PERMUTATION($S, pos - 1$)

SWAP(S, i, pos)

end for

end if

Dado un conjunto S de números enteros positivos, encontrar los subconjuntos que sumen la cantidad C . Si no se encuentra algún subconjunto que cumpla, se debe responder con el conjunto vacío.

Procedure 5 SUBSET_SUM

Input: A : Array, S : Set, $acum$: Integer, $target$: Integer, $level$: Integer

if $acum = target$ then

 print *solution*

end if

if $acum > target$ then

 return

end if

if $level < A.length$ then

 SUBSET_SUM($A, S, acum, target, level + 1$)

$S.ININSERT(A[level])$

 SUBSET_SUM($A, S, acum + A[level], target, level + 1$)

end if

524 Prime Ring Problem

A ring is composed of n (even number) circles as shown in diagram. Put natural numbers $1, 2, \dots, n$ into each circle separately, and the sum of numbers in two adjacent circles should be a prime.

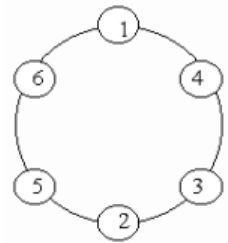
Note: the number of first circle should always be 1.

Input

n ($0 < n \leq 16$)

Output

The output format is shown as sample below. Each row represents a series of circle numbers in the ring beginning from 1 clockwise and anticlockwise. The order of numbers must satisfy the above requirements.



- 1 You are to write a program that completes above process.

¹<https://onlinejudge.org/external/5/524.pdf>

Sample Input

6

8

Sample Output

Case 1:

1 4 3 2 5 6

1 6 5 2 3 4

Case 2:

1 2 3 8 5 6 7 4

1 2 5 8 3 4 7 6

1 4 7 6 5 8 3 2

1 6 7 4 3 8 5 2