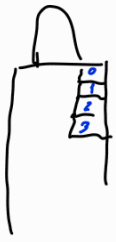
Búsqueda Completa

→ Buscar todas las soluciones posibles.

→ Generar todas las posibles salidas de una solución y escojo la mejor.



1000 combinaciones para un candado de 3 dígitos.

```
for i in (0...9)
    for j in (0..9)
        for k in (0..9):
            for l in (0...9)
                Hack(100·i+10·j+k+1000·l)
```

0, 1, 2, 3, 4, ..., 9, a, b, c..., z

```
for i in (0.... 10000)
    Hack(i)
```

```
d = [0, 1, ..., 9, a, b, ... z]
for l in d:
    for i in d:
        for j in d:
            for K in d:
                Hack(i+j+K+l)
```

```
global target_sz = 4
def Generate(w, sz)
    if (sz == target_size)
        Hack(w)
    for c in d:
        Generate(w+c, sz+1)
```

$O(|d|^{target\_size})$

Recursión

Caso base.

int int → Potencias con exponentes positivos

Pow(a, b) → Calcula a^b o a^b

```
if (b==0) return a.
if(b<0) return -1. // no esta definido aun.
return a* pow(a,b-1)
```

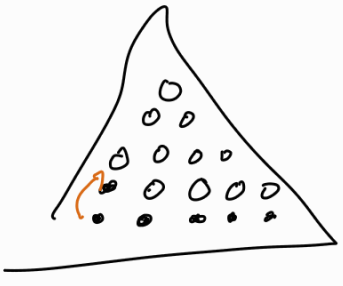
$\Rightarrow a^b = a \cdot a^{b-1}$

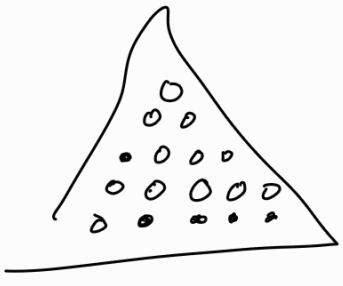
Pow(3, 2) → 3* pow(3,1) → 3* pow(3,0) → 3* pow(3,-1) .....

Pow(5, -3)

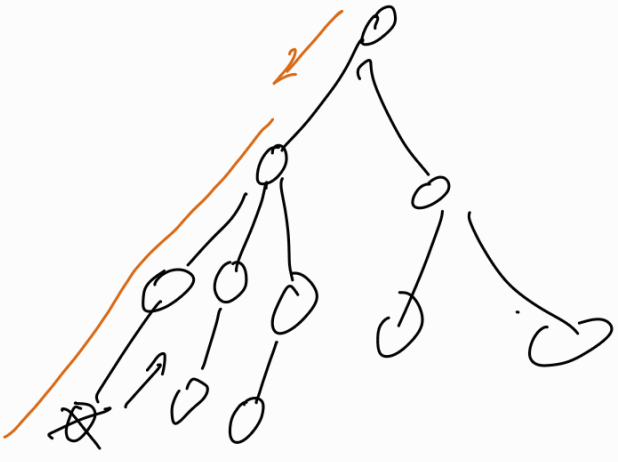
Busqueda de soluciones de juegos.

- Peg
- Cubo de rubik



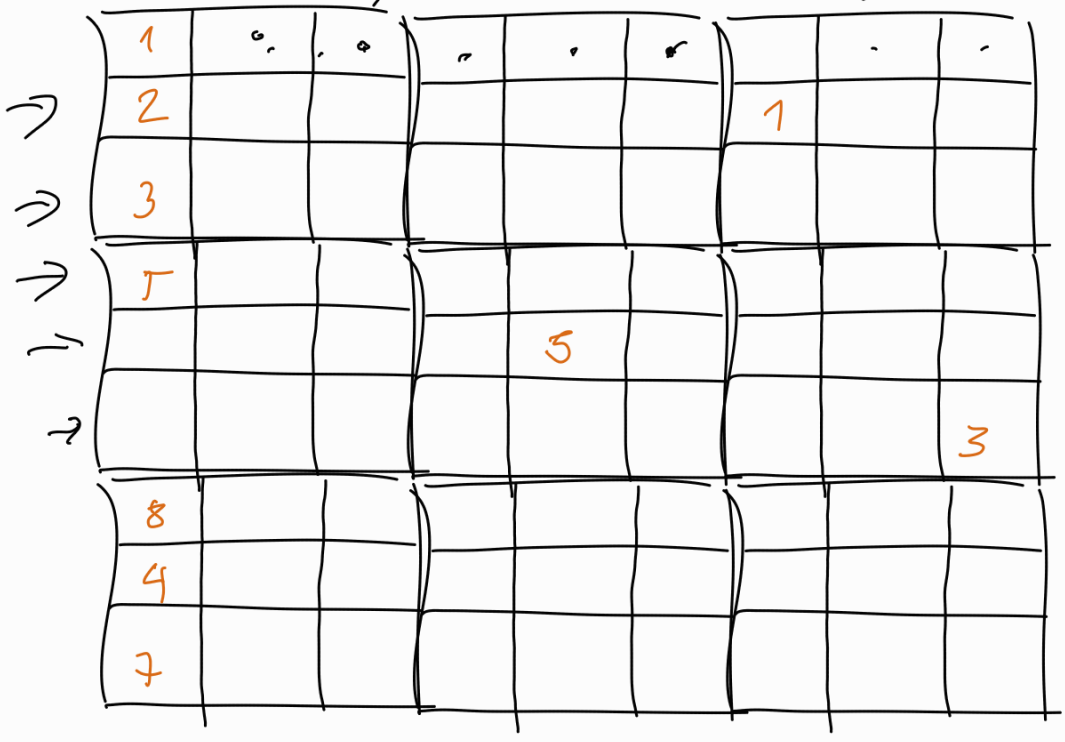


- Sudoku
- Crucigrama
- ⋮
- Ajedrez. → Heuristicas



En cada sub-cuadro, fila y columna deben aparecer los números del 1 al 9 sin repetirse.

En una misma fila, columna o sub-cuadro no puede aparecer el mismo número 2 veces.



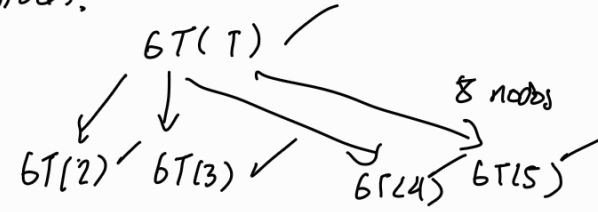
- Algunas casillas vienen con números asignados..
  → NO se pueden modificar.

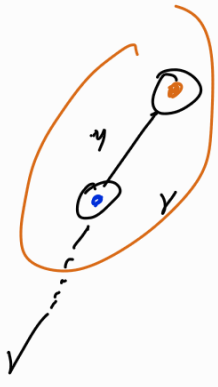
- Validar Tablero() → lógica que verifica si es valido.
- Generar Tablero (Tablero) {
  Priorizo casillas con menos opciones validas.

GT(T)

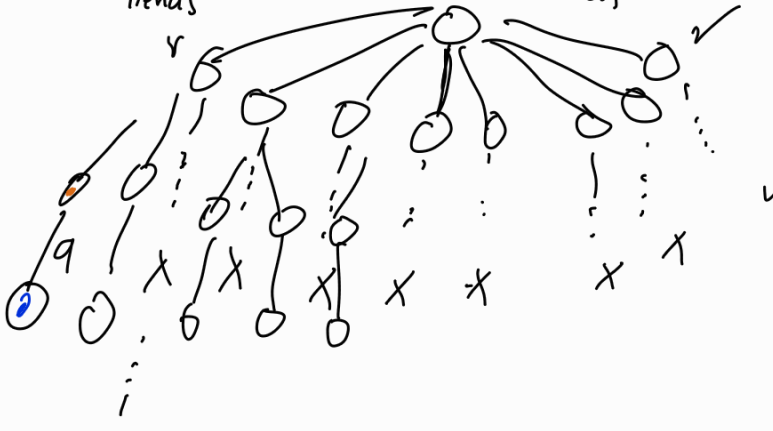
(1 nodo que explorar)

GT(T) // [5][0] → 9





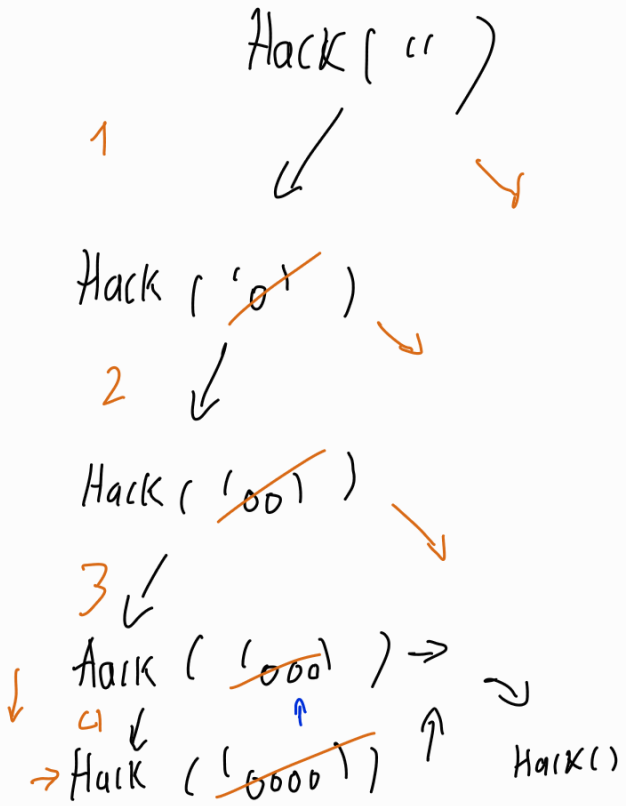
Cuando tengo todas las casillas llenas → revisu si hay solución



Backtracking

Memoria de stack

call stack




```
a = [0, ... 9]
Hack (word, n):
    if (n == 4): Attack(word)
    for c in a:
        Hack (word + c, n+1)
```

Op1: Hack ("")

Op2:

```
word = ""
Hack (word)
```

Op3:

```
global word = ""
Hack (n):
    if (n == 4): Attack()
    for c in a:
        word[n] = c
        Hack (n+1)
        word[n] = ""
```

Forward

Backtracking