Práctica 1: Técnicas Algorítmicas

Compilado: 21 de marzo de 2022

Backtracking

1. En este ejercicio vamos a resolver el problema de suma de subconjuntos, visto en la teórica, con la técnica de *backtracking*. A diferencia de la clase teórica, vamos a usar la representación binaria. Dado un multiconjunto $C = \{c_1, \ldots, c_n\}$ de números naturales y un natural k, queremos determinar si existe un subconjunto de C cuya sumatoria sea k. Notar que, a diferencia de la teórica, no necesitamos suponer que los elementos de C son todos distintos. Sí vamos a utilizar fuertemente que C está ordenado de alguna forma arbitraria pero conocida (i.e., C está implementado como la secuencia $c_1, \ldots, c_n$ o, análogamente, tenemos un iterador de C). Como se discute en la teórica, las *soluciones (candidatas)* son los vectores $a = (a_1, \ldots, a_n)$ de valores binarios; el subconjunto de C representado por a contiene a c_i si y sólo si $a_i = 1$. Luego, a es una solución *válida* cuando $\sum_{i=1}^{n} a_i c_i = k$. Asimismo, una *solución parcial* es un vector $p = (a_1, \ldots, a_i)$ de números binarios con $0 \leq i \leq n$. Si $i < n$, las soluciones *sucesoras* de p son $p \oplus 0$ y $p \oplus 1$, donde $\oplus$ indica la concatenación.

1 ?

a) Escribir el conjunto de soluciones candidatas para $C = \{6, 12, 6\}$ y $k = 12$.

b) Escribir el conjunto de soluciones válidas para $C = \{6, 12, 6\}$ y $k = 12$.

c) Escribir el conjunto de soluciones parciales para $C = \{6, 12, 6\}$ y $k = 12$.

d) Dibujar el árbol de *backtracking* correspondiente al algoritmo descrito arriba para $C = \{6, 12, 6\}$ y $k = 12$, indicando claramente la relación entre las distintas componentes del árbol y los conjuntos de los incisos anteriores.

a) $\Omega = \{ (0,0,0), \ \vdots \ , \ (1,1,1) \}$

$\#\Omega = 2^{\#c}$

```
algoritmo BT(a, k)
    si a es solución entonces
        procesar(a)
        retornar
    sino
        para cada a' ∈ Sucesores(a, k)
            BT(a', k + 1)
        fin para
    fin si
    retornar
```

b) $S = \{a, b\}$

$a = (1, 0, 1)$

$b = (0, 1, 0)$

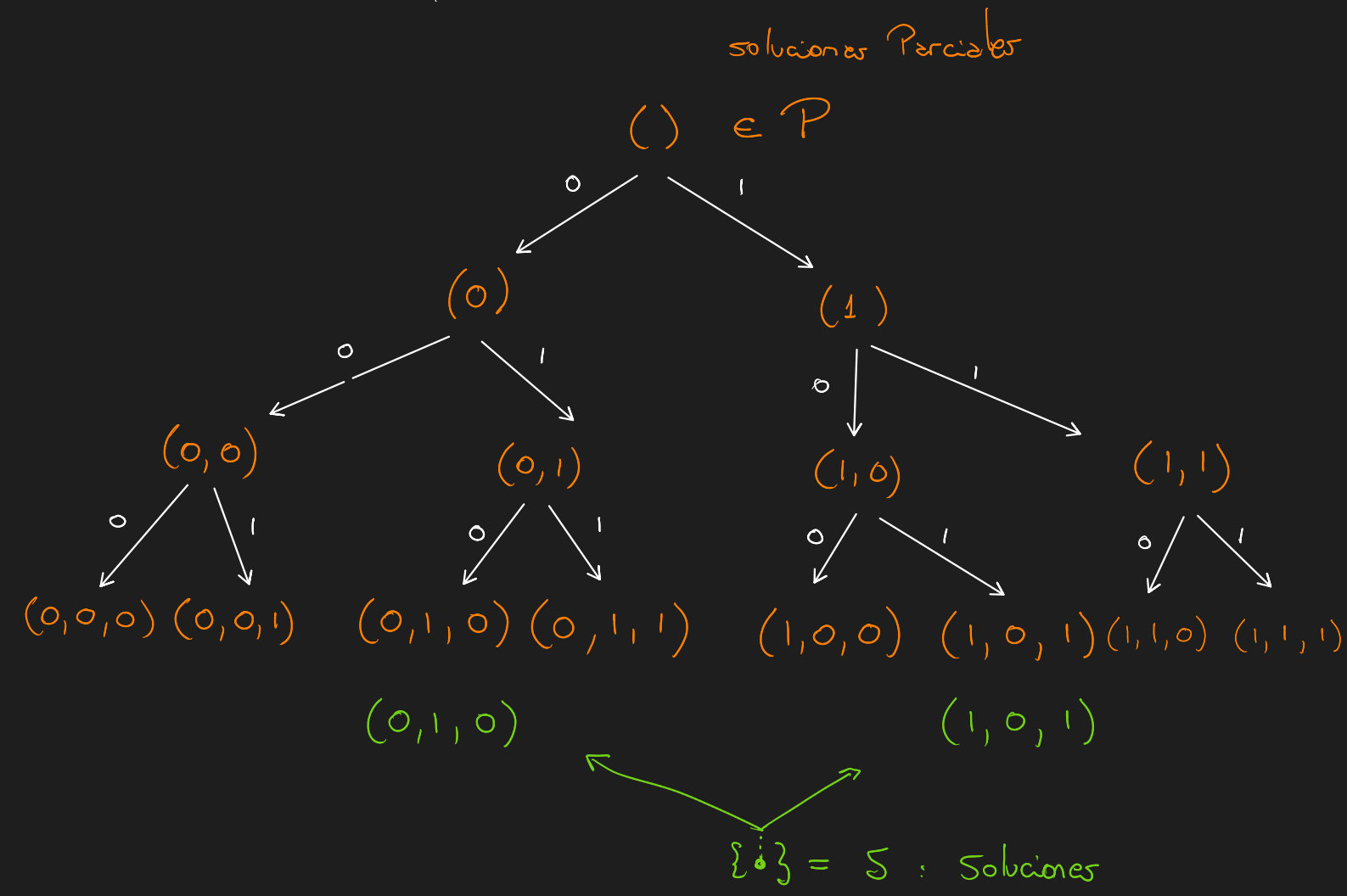
c) $P = \{ (), (0), (1),$

↖ arreglo vacío

$(1,0), (0,1),$

$(0,0), a, b ? \}$

d) $\{6, 12, 6\}$



e) Sea $\mathcal{C}$ la familia de todos los multiconjuntos de números naturales. Considerar la siguiente función recursiva ss: $\mathcal{C} \times \mathbb{N} \to \{V, F\}$ (donde $\mathbb{N} = \{0, 1, 2, \dots\}$, V indica verdadero y F falso):

$$\text{ss}(\{c_1, \dots, c_n\}, k) = \begin{cases} k = 0 & \text{si } n = 0 \\ \text{ss}(\{c_1, \dots, c_{n-1}\}, k) \vee \text{ss}(\{c_1, \dots, c_{n-1}\}, k - c_n) & \text{si } n > 0 \end{cases}$$

Convencerse de que $\text{ss}(C, k) = V$ si y sólo si el problema de subconjuntos tiene una solución válida para la entrada C, k. Para ello, observar que hay dos posibilidades para una solución válida $a = (a_1, \dots, a_n)$ para el caso $n > 0$: o bien $a_n = 0$ o bien $a_n = 1$. En el primer caso, existe un subconjunto de $\{c_1, \dots, c_{n-1}\}$ que suma k; en el segundo, existe un subconjunto de $\{c_1, \dots, c_{n-1}\}$ que suma $k - c_n$.

Si $a_n = 0$

f) Convencerse de que la siguiente es una implementación recursiva de ss en un lenguaje imperativo y de que retorna la solución para C, k cuando se llama con $C, |C|, k$. ¿Cuál es su complejidad?

1) subset_sum(C, i, j): // implementa ss($\{c_1, \dots, c_i\}, j$)
2) Si $i = 0$, retornar $(j = 0)$
3) Si no, retornar subset_sum($C, i - 1, j$) $\vee$ subset_sum($C, i - 1, j - C[i]$)

C |C| k

$(\{6, 12, 6\}, 3, 12)$

1)
3) $(C, 2, 12) \vee (C, 2, 12 - 6)$

I II

1)$_I$

3) $(C, 1, 12) \vee (C, 1, 12 - 12)$ → ret true

1)$_{II}$ $\vee$

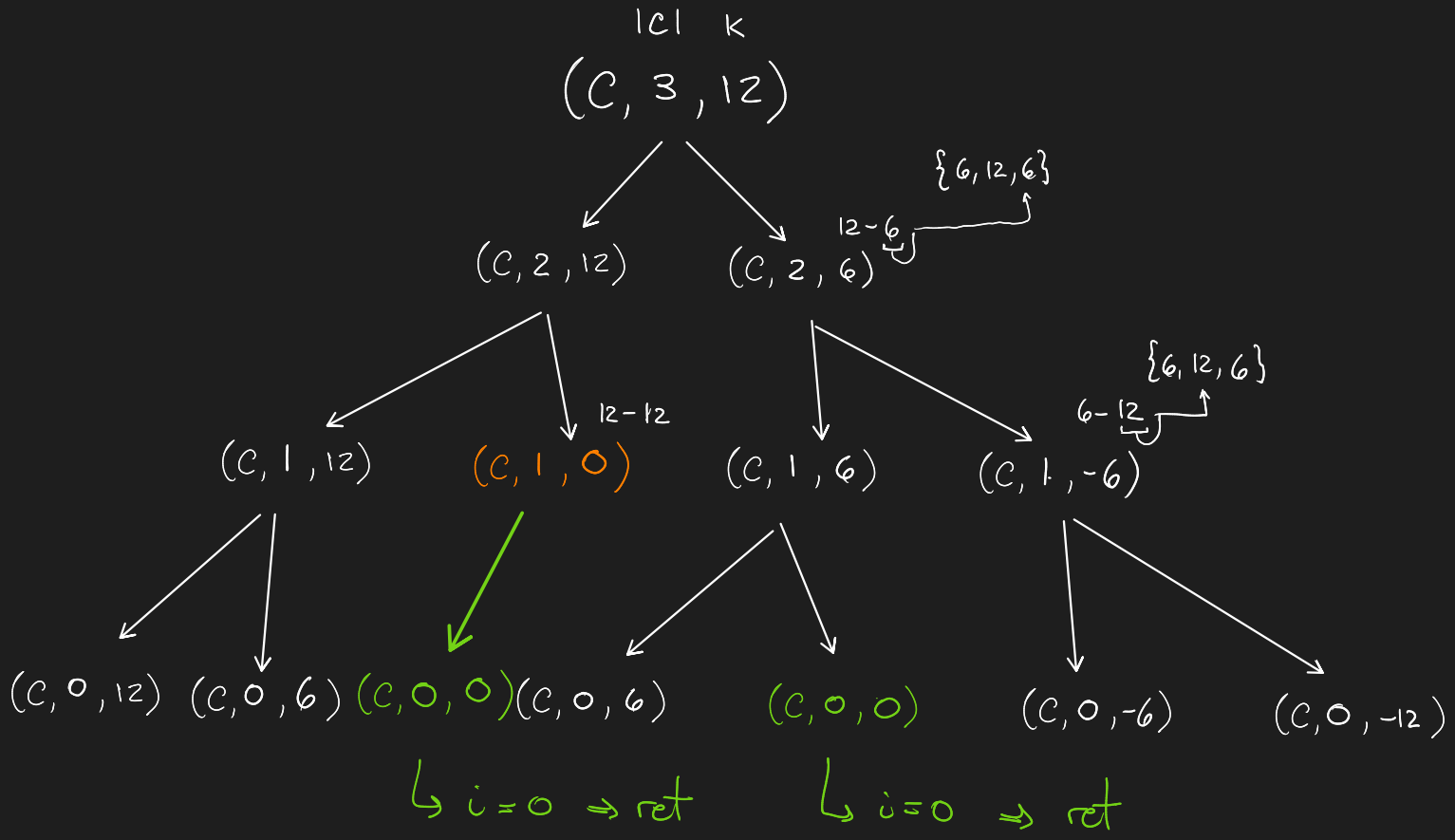
$(C, 1, 6) \vee (C, 1, 6 - 12)$

1) 3) $(C, 0, 12) \vee (C, 0, 12-6)$

$\vee (C, 0, 6) \vee (C, 0, -6-6)$

$\vee (C, 0, -6) \vee (C, 0, -6-6)$

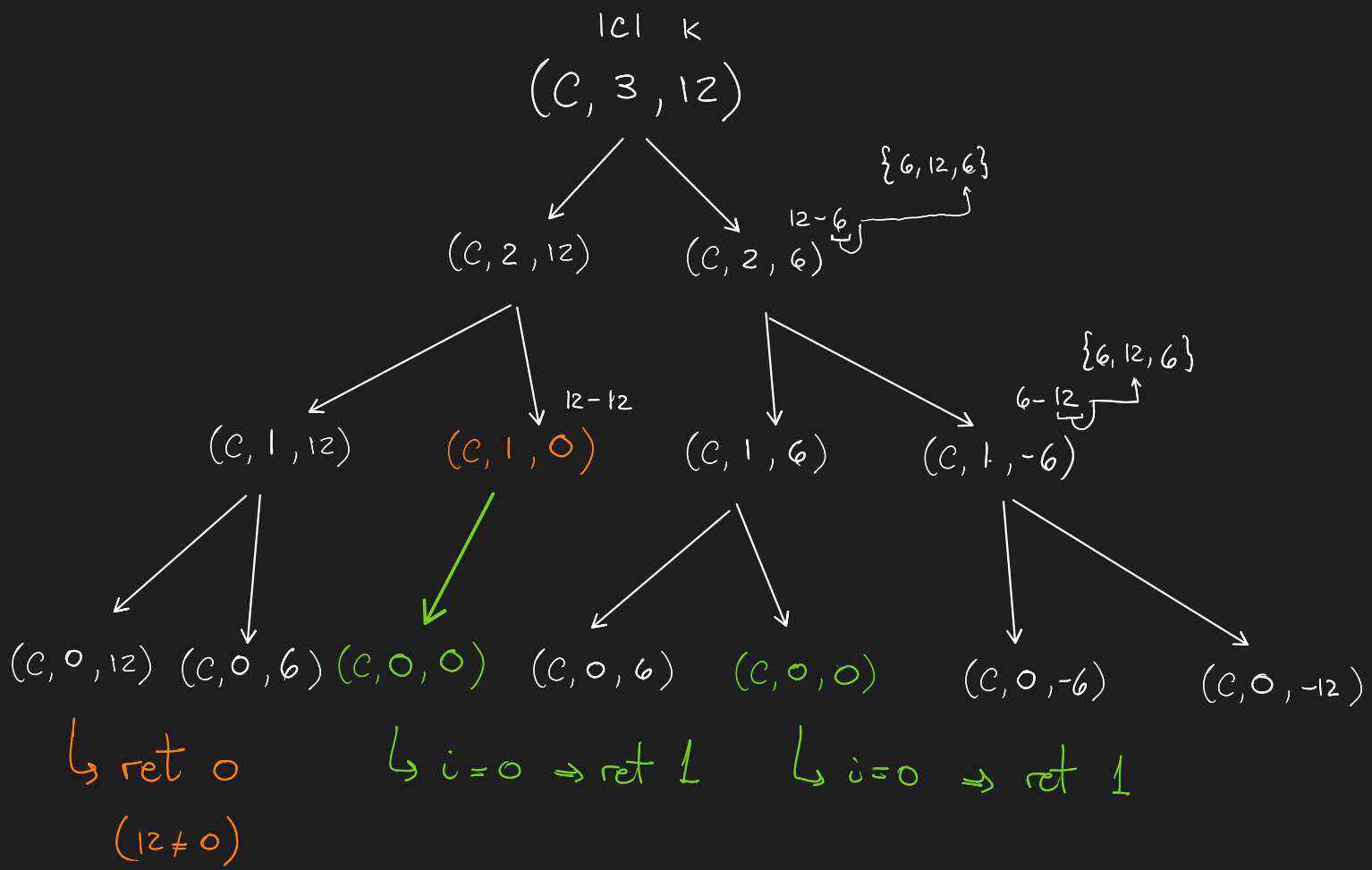
g) Dibujar el árbol de llamadas recursivas para la entrada $C = \{6, 12, 6\}$ y $k = 12$, y compararlo con el árbol de *backtracking*.



h) Considerar la siguiente *regla de factibilidad*: $p = (a_1, \ldots, a_i)$ se puede extender a una solución válida sólo si $\sum_{q=1}^{i} a_q c_q \leq k$. Convencerse de que la siguiente implementación incluye la regla de factibilidad.

1) subset_sum(C, i, j): // implementa ss$(\{c_1, \ldots, c_i\}, j)$
2) Si $j < 0$, retornar **falso** // regla de factibilidad
3) Si $i = 0$, retornar $(j = 0)$
4) Si no, retornar subset_sum$(C, i-1, j) \vee$ subset_sum$(C, i-1, j - C[i])$

Si j es negativo, quiere decir que sumé de más, por lo que sumando más números positivos no puedo "volver atrás" para obtener un resultado menor (el j original).



i) Definir otra regla de factibilidad, mostrando que la misma es correcta; no es necesario implementarla.

Puedo restar cada valor elegido a la suma total (precalc.)

```
int s = sum(C)
...
if
```

j) Modificar la implementación para imprimir el subconjunto de C que suma k, si existe. **Ayuda:** mantenga un vector con la solución parcial p al que se le agregan y sacan los elementos en cada llamada recursiva; tenga en cuenta de no suponer que este vector se copia en cada llamada recursiva, porque cambia la complejidad.

(C, i, j, p)

1) subset_sum(C, i, j): // implementa ss$(\{c_1, \ldots, c_i\}, j)$

2) Si $j < 0$, retornar **falso** // regla de factibilidad

3) Si $i = 0$, retornar ~~(0,0)~~
   ↳ p

4) Si no, retornar subset_sum$(C, i-1, j) \vee$ subset_sum$(C, i-1, j - C[i])$

$(j=0) \cdot (C, i-1, j, p \oplus 0) + ((j - C[i]) = 0) \cdot (C, i-1, j - C[i], p \oplus 1)$

True = 1

escalar

vector

p inicial = ( ) ← arreglo vacío

Siempre llega al final de cada rama antes de volver, por los que los valores de p nunca se pisan y siempre que se agregan, se vuelve atrás para volver a agregar otros.

2. Un *cuadrado mágico de orden* n, es un cuadrado con los números $\{1, \ldots, n^2\}$, tal que todas sus filas, columnas y las dos diagonales suman lo mismo (ver figura). El número que suma cada fila es llamado *número mágico*.

2	7	6
9	5	1
4	3	8

Existen muchos métodos para generar cuadrados mágicos. El objetivo de este ejercicio es contar cuántos cuadrados mágicos de orden n existen.

a) ¿Cuántos cuadrados habría que generar para encontrar todos los cuadrados mágicos si se utiliza una solución de fuerza bruta?

b) Enunciar un algoritmo que use *backtracking* para resolver este problema que se base en la siguientes ideas:

- La solución parcial tiene los valores de las primeras $i-1$ filas establecidos, al igual que los valores de las primeras j columnas de la fila i.
- Para establecer el valor de la posición $(i, j+1)$ (o $(i+1, 1)$ si $j = n$ e $i \neq n$) se consideran todos los valores que aún no se encuentran en el cuadrado. Para cada valor posible, se establece dicho valor en la posición y se cuentan todos los cuadrados mágicos con esta nueva solución parcial.

Mostrar los primeros dos niveles del árbol de *backtracking* para $n = 3$.

c) Demostrar que el árbol de *backtracking* tiene $\mathcal{O}((n^2)!)$ nodos en peor caso.

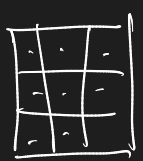
d) Considere la siguiente poda al árbol de *backtracking*: al momento de elegir el valor de una nueva posición, verificar que la suma parcial de la fila no supere el número mágico. Verificar también que la suma parcial de los valores de las columnas no supere el número mágico. Introducir estas podas al algoritmo e implementarlo en la computadora. ¿Puede mejorar estas podas?

e) Demostrar que el número mágico de un cuadrado mágico de orden n es siempre $(n^3 + n)/2$. Adaptar la poda del algoritmo del ítem anterior para que tenga en cuenta esta nueva información. Modificar la implementación y comparar los tiempos obtenidos para calcular la cantidad de cuadrados mágicos.

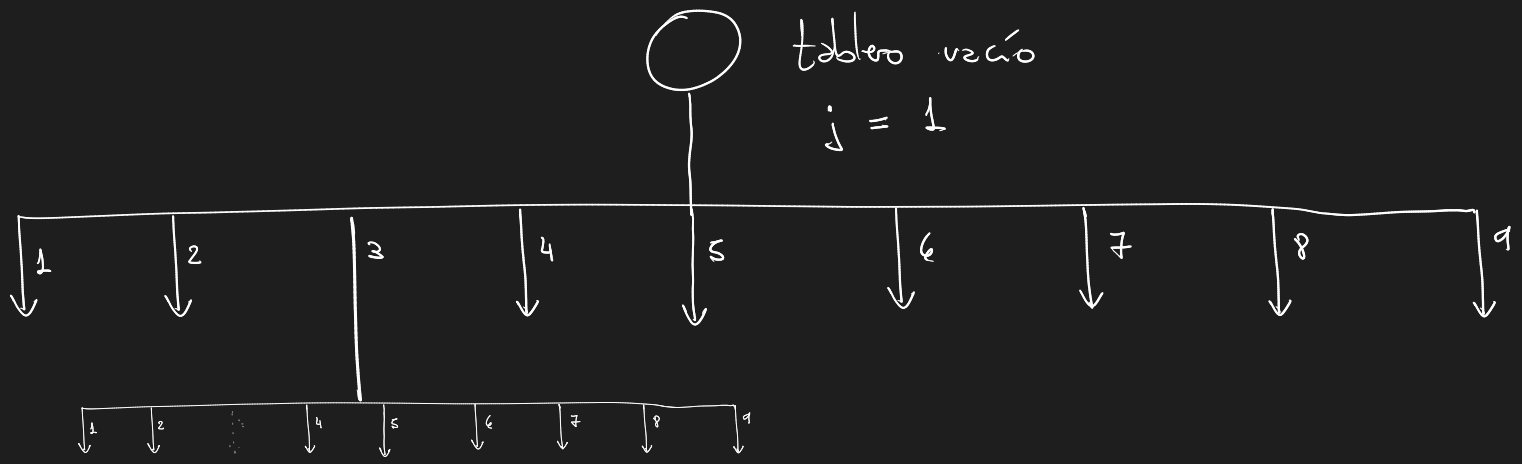
a) $(n^2)!$

b) vector tablero = $[[1, 2 \ldots, n], \ldots, [n^2-n+1, \ldots, n^2]]$ $\quad C = [1, \ldots, n^2]$

```
BT ( tablero, C, j)
  if j == n^2 + 1
     return true
  a = j // 3        # div. entera
  b = j % 3 + 1     # resto
  for i = j to n^2
     tablero[a][b] = C[i]
     BT ( tablero , C , i + 1)
```



llamo como
BT (tablero, C, 1)



c) Para el 1° casillero tengo n^2 posibilidades

Para el 2° casillero tengo $n^2 - 1$ posibilidades (pues use 1 en el 1°)

⋮

Para el último casillero tengo solo 1 posibilidad

∴ $(n^2)!$ nodos.

d)
```
int M = numero_magico (n)
BT (    )
    :
    a = j // 3          # div. entera
    b = j % 3 + 1       # resto
    if   sum ( tablero [1:a][1:b] ) > M
         return
```

Se puede mejorar agregando podas a filas/columnas completas que sumen un valor DISTINTO al número mágico correspondiente.

e) $f(n) = \frac{n^3 + n}{2}$

$n = 1 \Rightarrow f(1) = 1$ ✓

$n = 2 \Rightarrow f(2) = 5$ ✓

HI : $f(n) = \frac{n^3 + n}{2}$

q.q : $f(n+1) = \frac{(n+1)^3 + n + 1}{2}$

$$\frac{(n+1)^3 + n + 1}{2} = \frac{n^3 + 3n^2 + 3n + 1 + n + 1}{2}$$

$$f(n+1) = \underbrace{\frac{n^3 + n}{2}}_{f(n)} + \frac{3n^2 + 3n + 1}{2}$$

no me sirve esto :(

3. Dada una matriz simétrica M de $n \times n$ números naturales y un número k, queremos encontrar un subconjunto I de $\{1, \ldots, n\}$ con $|I| = k$ que maximice $\sum_{i,j \in I} M_{ij}$. Por ejemplo, si $k = 3$ y:

$$M = \begin{pmatrix} 0 & 10 & 10 & 1 \\ - & 0 & 5 & 2 \\ - & - & 0 & 1 \\ - & - & - & - \end{pmatrix},$$

entonces $I = \{1, 2, 3\}$ es una solución óptima.

suma de todo "el cuadrado" de valores?

a) Diseñar un algoritmo de *backtracking* para resolver el problema, indicando claramente cómo se codifica una solución candidata, cuáles soluciones son válidas y qué valor tienen, qué es una solución parcial y cómo se extiende cada solución parcial.

```
BT(M, I, k, s
    if |I| == k
        return
```