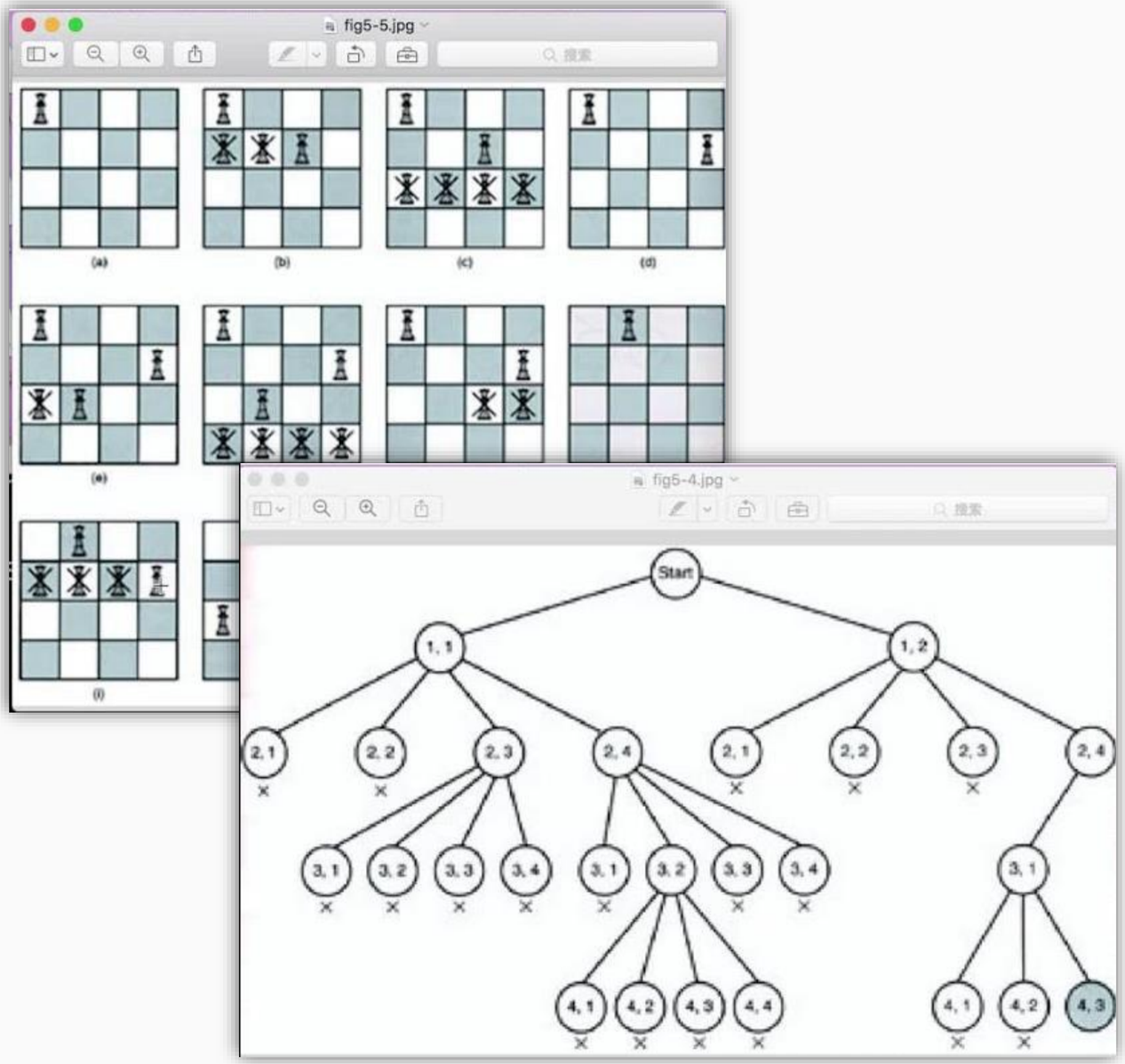
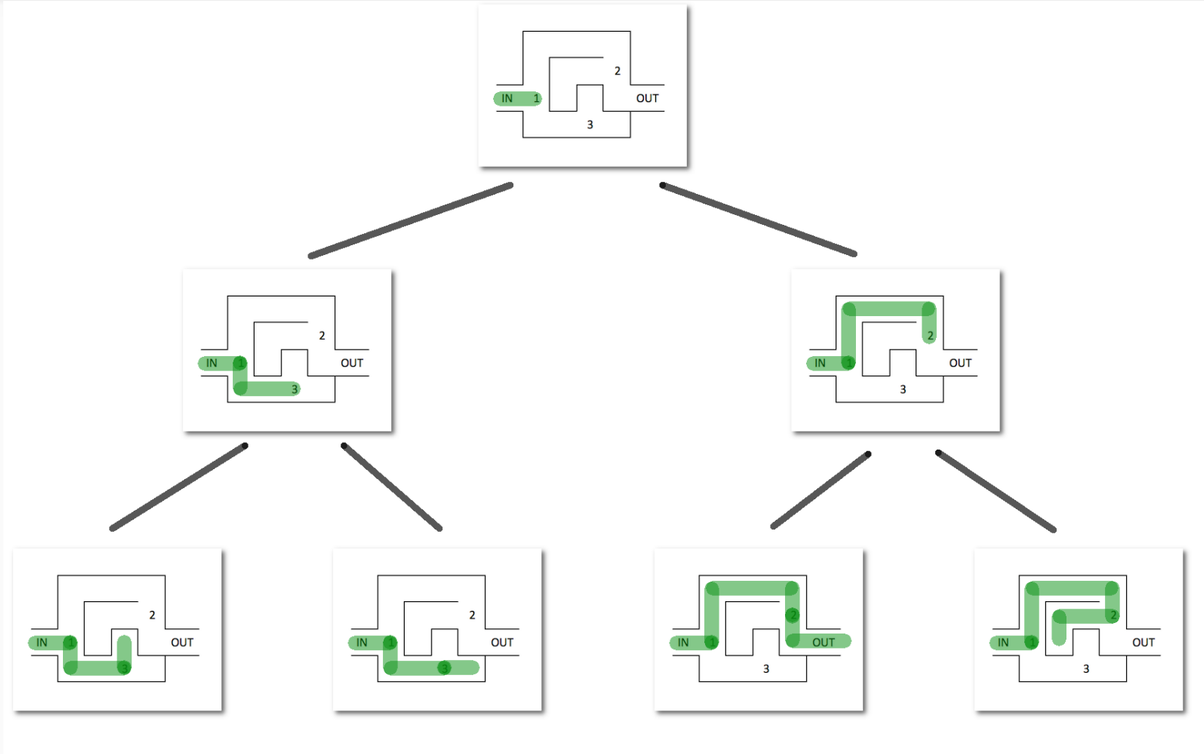
UTEC
UNIVERSIDAD DE INGENIERÍA
Y TECNOLOGÍA

Welcome to Algorithms and Data Structures! - CS2100

Backtracking

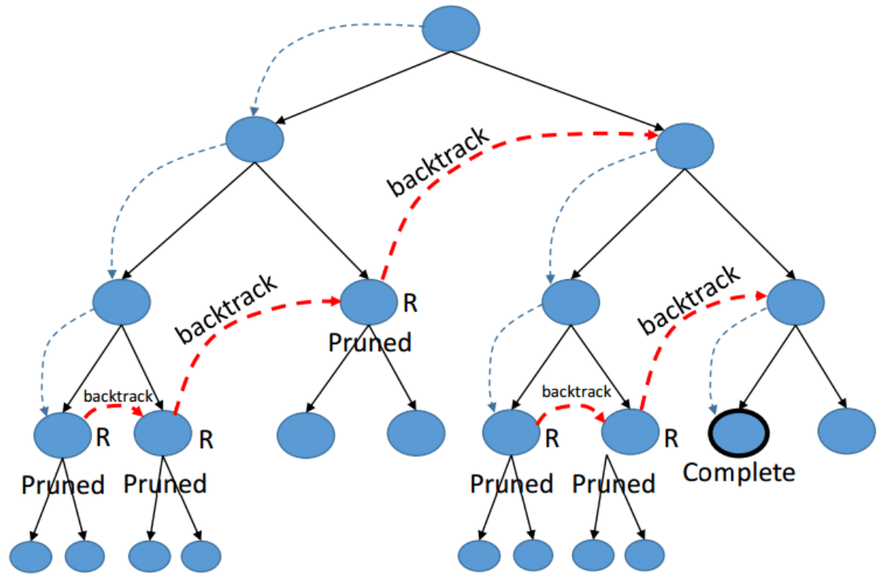




Backtracking

- El **backtracking** es una técnica general de resolución de problemas, aplicable en problemas de **optimización**, juegos y otros tipos.
- El backtracking realiza una **búsqueda exhaustiva** y **sistemática** en el espacio de soluciones. Por ello, suele resultar muy ineficiente.

```
procedure EXPLORE(node n)
    if REJECT(n) then return
    if COMPLETE(n) then
        OUTPUT(n)
    for n_i : CHILDREN(n) do EXPLORE(n_i)
```

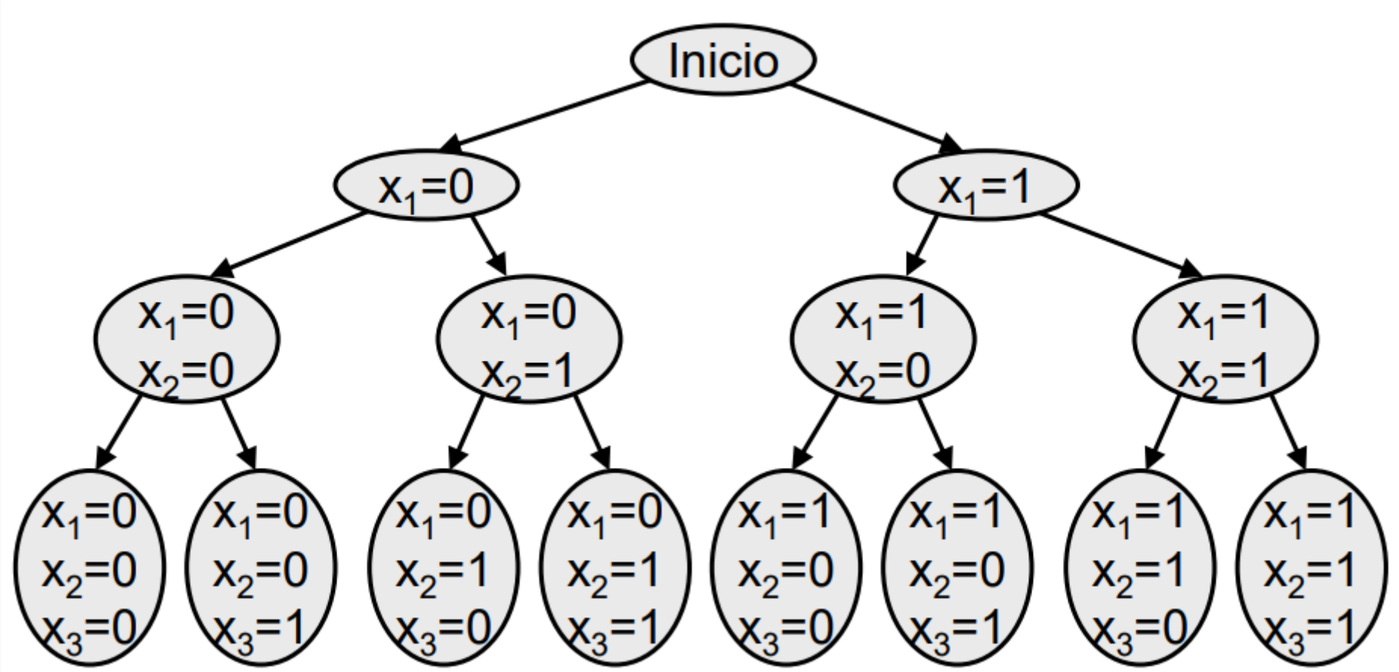


Backtracking: Método General

- Una solución se puede expresar como una tupla, (x_1 , x_2 , ..., x_n), que satisfaga unas restricciones y tal vez que optimice cierta función objetivo
- En cada momento, el algoritmo se encontrará en cierto nivel k, con una solución parcial (x_1 , ..., x_k).
  - Si se puede añadir un nuevo elemento a la solución x_{k+1}, se genera y se avanza al nivel k+1.
  - Si no, se prueban otros valores para x_k .
  - Si no existe ningún valor posible por probar, entonces se retrocede al nivel anterior k-1.
  - Se sigue hasta que la solución parcial sea una solución completa del problema, o hasta que no queden más posibilidades por probar.

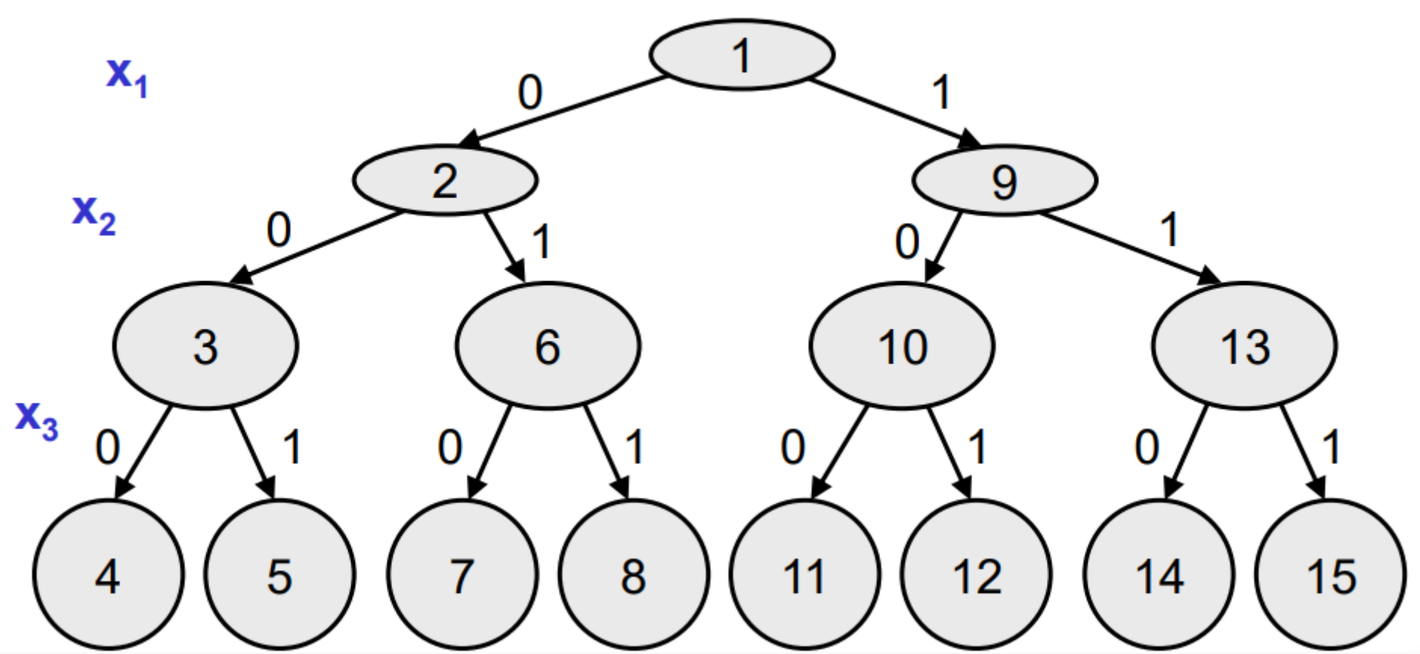
Backtracking: Método General

- El resultado es equivalente a hacer un recorrido en profundidad en el árbol de soluciones.



Backtracking: Método General

- Representación simplificada del árbol.



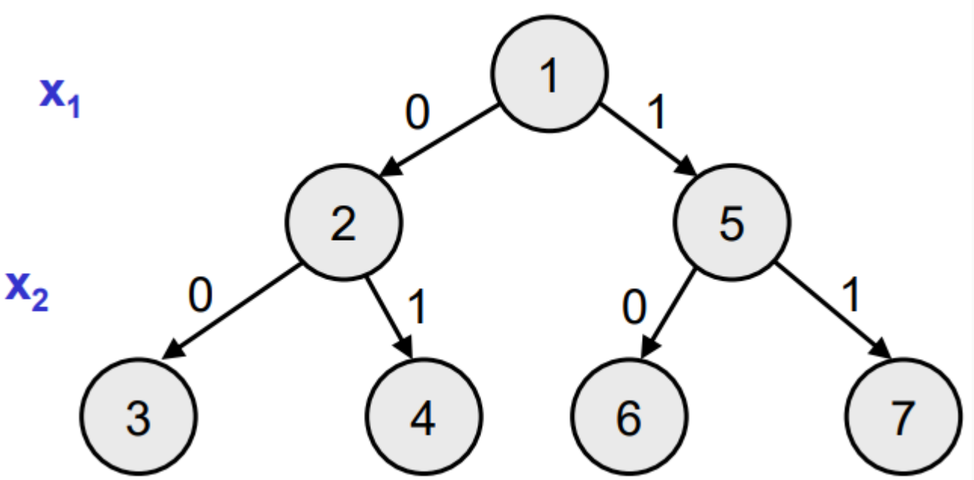
Backtracking: Método General

- **Árboles de backtracking:**
  - El árbol es simplemente una forma de representar la ejecución del algoritmo.
  - Es implícito, no almacenado (no necesariamente).
  - El recorrido es en profundidad, normalmente de izquierda a derecha.
  - La primera decisión para aplicar backtracking: ¿cómo es la forma del árbol?
  - Preguntas relacionadas: ¿Qué significa cada valor de la tupla solución (x1 , ..., xn)? ¿Cómo es la representación de la solución al problema?

Backtracking: Método General

- **Árboles binarios**: s= (x1 , x2 , ..., xn), con xi ϵ {0, 1}

- **Tipo de problemas:** elegir ciertos elementos de entre un conjunto, sin importar el orden de los elementos.
  - Encontrar un subconjunto de {12, 23, 1, 8, 33, 7, 22} que sume exactamente 50.
  - Problema de la mochila 0/1.



Backtracking: Método General

- **Algoritmo**: ¡esquema general!
- **Backtracking (var s: TuplaSolución)**

```
nivel:= 1
s:= s_INICIAL
fin:= false
repetir
        Generar (nivel, s)
        si Solución (nivel, s) entonces
                fin:= true
        sino si Criterio (nivel, s) entonces
                nivel:= nivel + 1
        sino
                mientras NOT MasHermanos (nivel, s) hacer
                        Retroceder (nivel, s)
        finsi
hasta fin
```

Backtracking:
Problema del Subconjunto

Backtracking: Problema del Subconjunto

- **Ejemplo de problema:** encontrar un subconjunto del conjunto T= {t1, t2, ..., tn } que sume exactamente P.
- **Variables:**
  - Representación de la solución con un árbol binario.
  - **S**: array [1..n] de {-1, 0, 1}
    - s[i] = 0 → el número i-ésimo no se utiliza
    - s[i] = 1 → el número i-ésimo sí se utiliza
    - s[i] = -1 → valor de inicialización (número i-ésimo no estudiado)
  - $\mathbf{S}_{\mathbf{INICIAL}}$: (-1, -1, ..., -1)
  - **fin**: valdrá **true** cuando se haya encontrado solución.
  - **tact**: suma acumulada hasta ahora (inicialmente 0).

Backtracking: Problema del Subconjunto

- **Funciones**:
  - **Generar (nivel, s)**

    s[nivel]:= s[nivel] + 1

    si s[nivel] == 1 **entonces** tact:= tact + tnivel
  - **Solución (nivel, s)**

    devolver (nivel == n) Y (tact == P)
  - **Criterio (nivel, s)**

    devolver (nivel < n) Y (tact ≤ P)
  - **MasHermanos (nivel, s)**

    devolver s[nivel] < 1
  - **Retroceder (nivel, s)**

    tact:= tact – t_{nivel}*s[nivel]

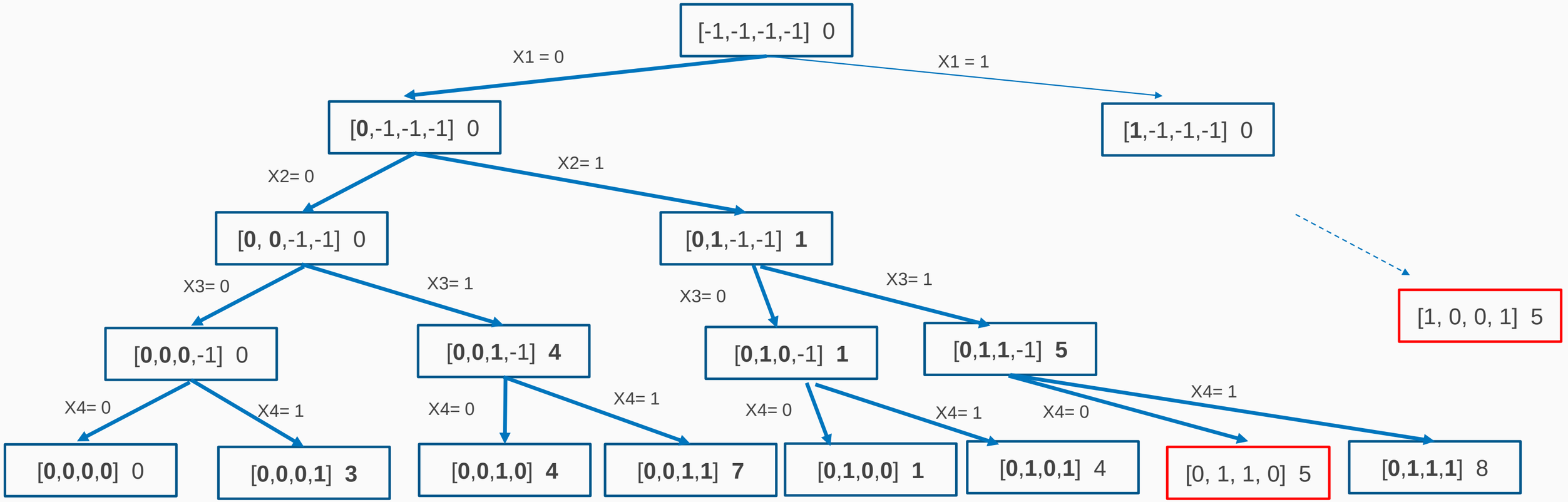
    s[nivel]:= -1

    nivel:= nivel – 1

```
Backtracking (var s: TuplaSolución)
    nivel:= 1
    s:= sINICIAL
    fin:= false
    repetir
        Generar (nivel, s)
        si Solución (nivel, s) entonces
            fin:= true
        sino si Criterio (nivel, s) entonces
            nivel:= nivel + 1
        sino
            mientras NOT MasHermanos (nivel, s) hacer
                Retroceder (nivel, s)
        finsi
    hasta fin
```

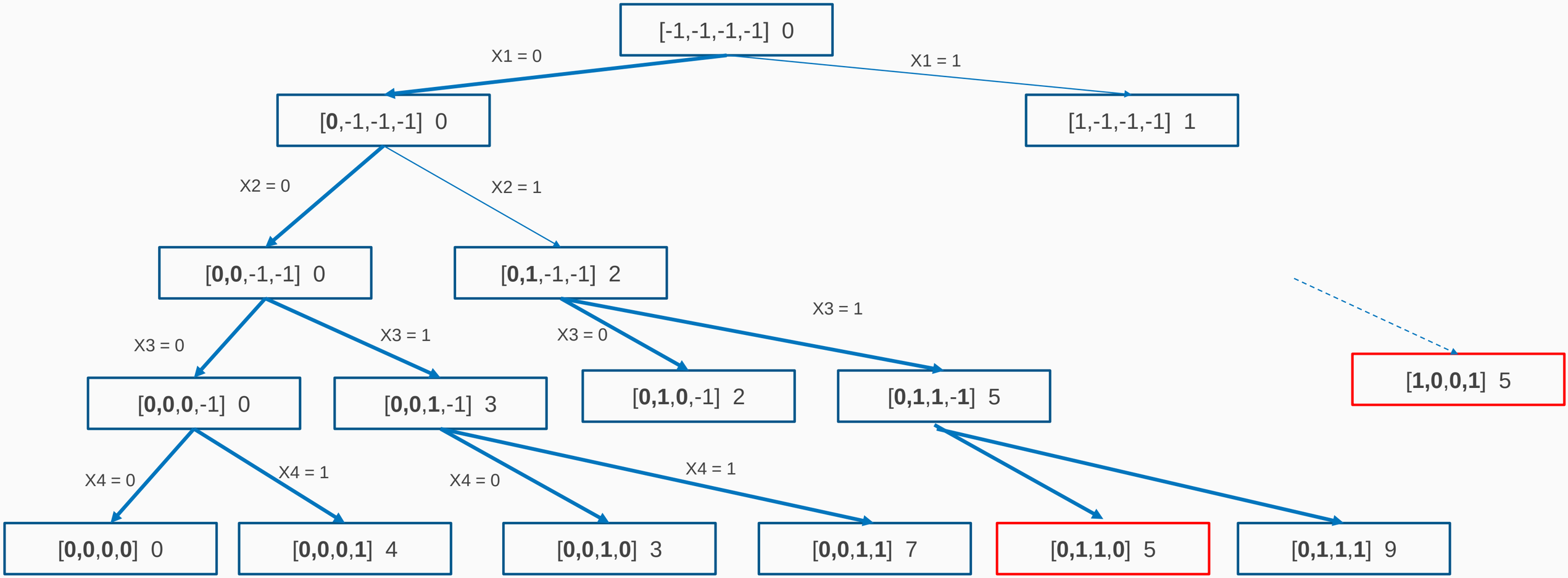
Backtracking: Problema del Subconjunto

Ejemplo: considere el conjunto {2, 1, 4, 3} se busca el subconjunto que sume 5



Backtracking: Problema del Subconjunto

Ejemplo: considere el conjunto {1, 2, 3, 4} se busca el subconjunto que sume 5



Backtracking: Variaciones

1. ¿Y si no es seguro que exista una solución?
2. ¿Y si queremos almacenar todas las soluciones (no solo una)?
3. ¿Y si el problema es de optimización (maximizar o minimizar)?
4. ¿Y si podamos el árbol de soluciones para evitar cálculos innecesarios?

```
Backtracking (var s: TuplaSolución)
    nivel:= 1
    s:= sINICIAL
    fin:= false
    repetir
            Generar (nivel, s)
            si Solución (nivel, s) entonces
                    fin:= true
            sino si Criterio (nivel, s) entonces
                    nivel:= nivel + 1
            sino
                    mientras NOT MasHermanos (nivel, s) hacer
                            Retroceder (nivel, s)
            finsi
    hasta fin
```

Backtracking: Método General

- **Caso 1:** Puede que no exista ninguna solución
- **Backtracking (var s: TuplaSolución)**

```
nivel:= 1
s:= s_INICIAL
fin:= false
repetir
        Generar (nivel, s)
        si Solución (nivel, s) entonces
                fin:= true
        sino si Criterio (nivel, s) entonces
                nivel:= nivel + 1
        sino
                mientras NOT MasHermanos (nivel, s) AND (nivel > 0) hacer
                        Retroceder (nivel, s)
        finsi
hasta fin OR (nivel == 0)
```

Backtracking: Método General

- **Caso 2:** Queremos almacenar todas las soluciones
- **Backtracking (var s: TuplaSolución)**

```
nivel:= 1
s:= s_INICIAL
repetir
        Generar (nivel, s)
        si Solución (nivel, s) entonces
                Almacenar(nivel, s)
        sino si Criterio (nivel, s) entonces
                nivel:= nivel + 1
        sino
                mientras NOT MasHermanos (nivel, s) AND (nivel > 0) hacer
                        Retroceder (nivel, s)
        finsi
hasta (nivel == 0)
```

Backtracking: Método General

- **Algoritmo:** Problema de optimización (maximización)
- **Backtracking (var s: TuplaSolución)**

```
nivel:= 1
s:= s_INICIAL
voa := -∞ ; soa := ∅
repetir
        Generar (nivel, s)
        si Solución (nivel, s) AND Valor(s)>voa entonces
                voa := Valor(s); soa = s
        sino si Criterio (nivel, s) entonces
                nivel:= nivel + 1
        sino
                mientras NOT MasHermanos (nivel, s) AND (nivel > 0) hacer
                        Retroceder (nivel, s)
        finsi
hasta fin OR (nivel == 0)
```

voa: valor optimo actual
soa: solución optima actual

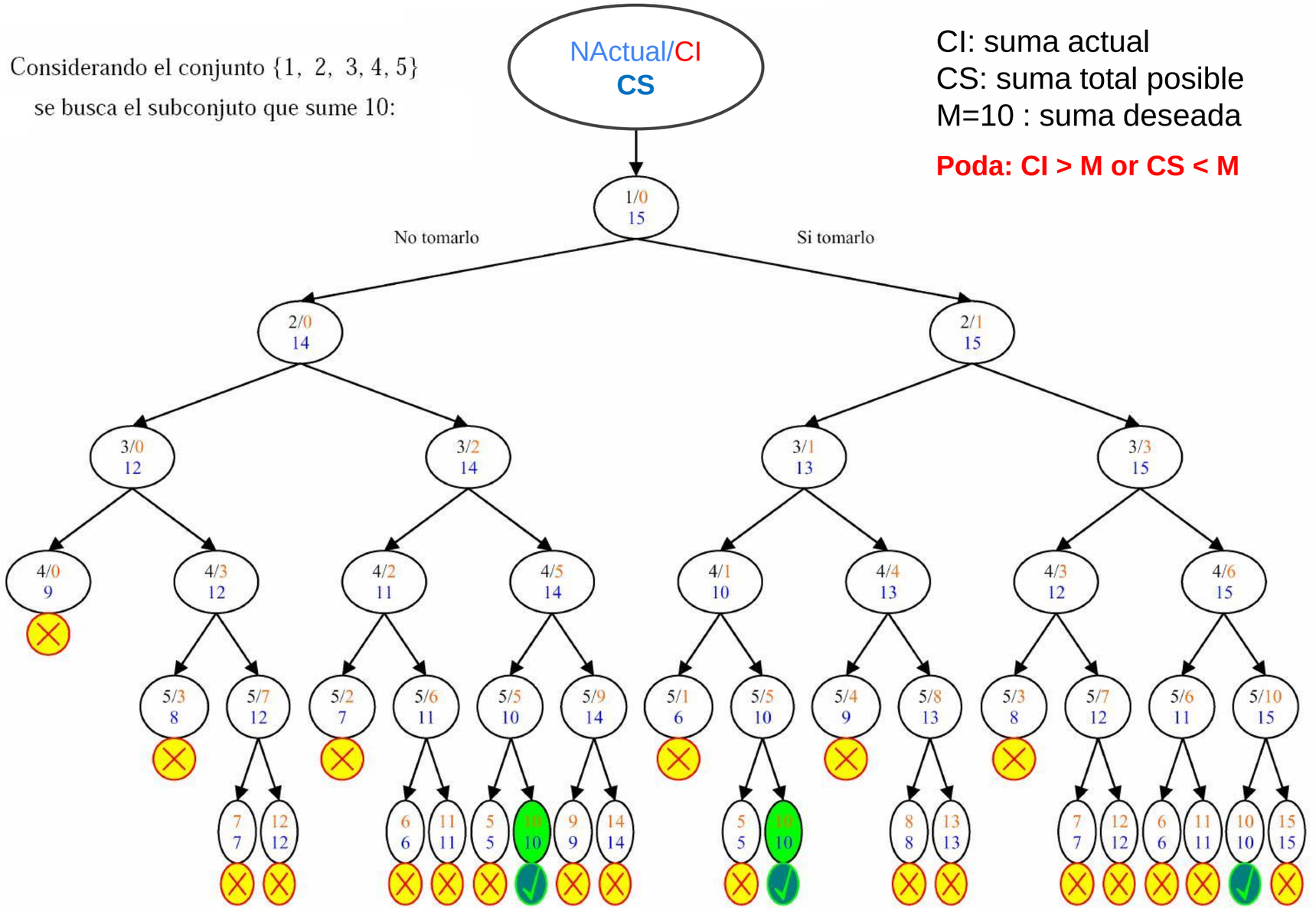
Backtracking: Ramificacion y Poda

- **Caso 4:** Ramificacion y Poda

- Suponer el mismo problema de encontrar todos los subconjuntos de un conjunto dado de enteros positivos {x1, x2..., xn} que sumen una cantidad M. Para resolverlo aplicando poda utilizamos lo siguiente:
  - CI: La cota inferior usada es la suma de los elementos de la solución actual.
  - CS: La cota superior es la cota inferior más los elementos que faltan por tratar.
  - Condicion de poda:
    - CI > M or CS < M

Backtracking: Ramificacion y Poda

Considerando el conjunto {1, 2, 3, 4, 5}
se busca el subconjuto que sume 10:



CI: suma actual
CS: suma total posible
M=10 : suma deseada

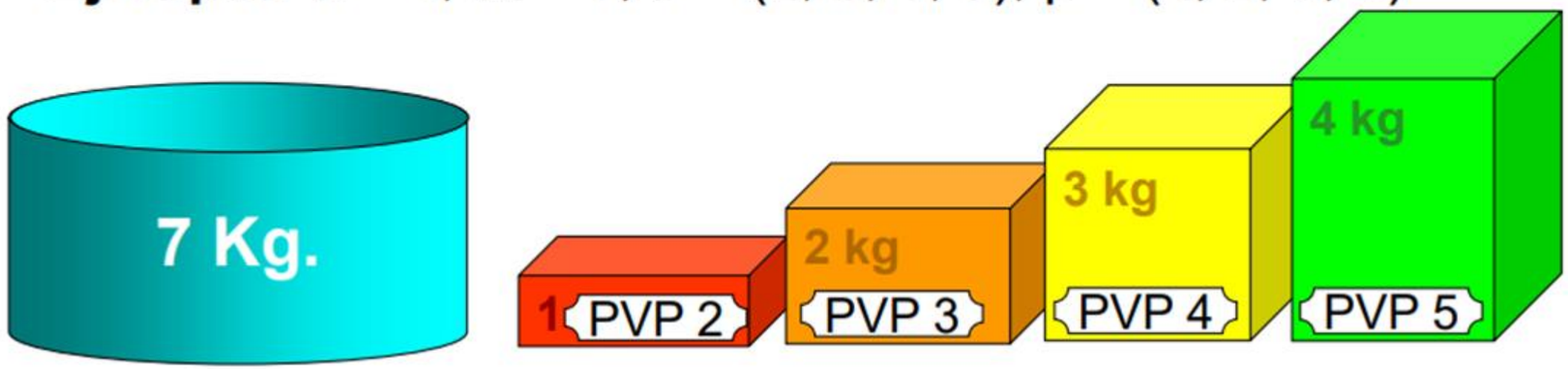
Poda: CI > M or CS < M

Backtracking:
Problema de la Mochila 0/1

Backtracking: Problema de la Mochila 0/1

- **Datos del problema:**
  - **n**: número de objetos disponibles.
  - **M**: capacidad de la mochila.
  - **p = (p_1, p_2, ..., p_n)** pesos de los objetos.
  - **b = (b_1, b_2, ..., b_n)** beneficios de los objetos.
- **Formulación matemática:**

  Maximizar $\sum_{i=1..n} x_i b_i$; sujeto a la restricción $\sum_{i=1..n} x_i p_i \leq M$, y $x_i \in \{0,1\}$
- **Ejemplo:** n = 4; M = 7; b = (2, 3, 4, 5); p = (1, 2, 3, 4)



Backtracking: Problema de la Mochila 0/1

- **Backtracking (var s: array[1..n] de enteros)**

```
nivel:= 1; s:= s_INICIAL
voa := -∞ ; soa := ∅
pact := 0; bact := 0
repetir
        Generar (nivel, s)
        si Solución (nivel, s) AND bact > voa entonces
                voa := bact;  soa = s
        sino si Criterio (nivel, s) entonces
                nivel:= nivel + 1
        sino
                mientras NOT MasHermanos (nivel, s) AND (nivel > 0) hacer
                        Retroceder (nivel, s)
        finsi
hasta fin OR (nivel == 0)
```

pact: peso actual
bact: beneficio actual

Backtracking: Problema de la Mochila 0/1

- **Funciones**:
  - **Generar (nivel, s)** → Probar primero 0 y luego 1

    s[nivel]:= s[nivel] + 1

    pact:= pact + p[nivel]*s[nivel]

    bact:= bact + b[nivel]*s[nivel]

```
si s[nivel]==1 entonces
        pact:= pact + p[nivel]
        bact:= bact + b[nivel]
finsi
```

  - **Solución (nivel, s)**

    devolver (nivel == n) AND (pact ≤ M)
  - **Criterio (nivel, s)**

    devolver (nivel < n) AND (pact ≤ M)
  - **MasHermanos (nivel, s)**

    devolver s[nivel] < 1
  - **Retroceder (nivel, s)**

    pact:= pact – p[nivel]*s[nivel]

    bact:= bact – b[nivel]*s[nivel]

    s[nivel]:= -1

    nivel:= nivel – 1

Backtracking: Problema de la Mochila 0/1

- **Ejemplo:** n = 4; M = 7; b = (2, 3, 4, 5); p = (1, 2, 3, 4)