Exploración exhaustiva y vuelta atrás

Acevedo Martinez Armando
Angeles Carcamo Omar Alejandrp
García Quiroz Gustavo Ivan

Programa

1.4 Exploración exhaustiva y vuelta atrás

- 1.4.1 Exploración exhaustiva
- 1.4.2 Programación por vuelta atrás
- 1.4.3 Nociones de complejidad de la exploración exhaustiva y vuelta atrás

Introducción

Muchos problemas de optimización solo pueden resolverse de manera exacta explorando todas las posibles combinaciones de elementos y tratando de encontrar aquellas que sean solución, eligiendo entre ellas una cualquiera o aquella que optimice una función. Ciertos problemas admiten estrategias de selección de elementos que conducen a soluciones optimas.

Aparece entonces la necesidad de volver atrás, es decir, deshacer decisiones previas que se ha demostrado que no conducían a una solución (por el contrario, los algoritmos voraces nunca deshacen una decisión previamente tomada).

Introducción

Se trata de un tipo particular de algoritmos de fuerza bruta que se emplean para problemas donde se busca un elemento con una propiedad especial, usualmente entre objetos combinatorios como permutaciones, combinaciones, o subconjuntos

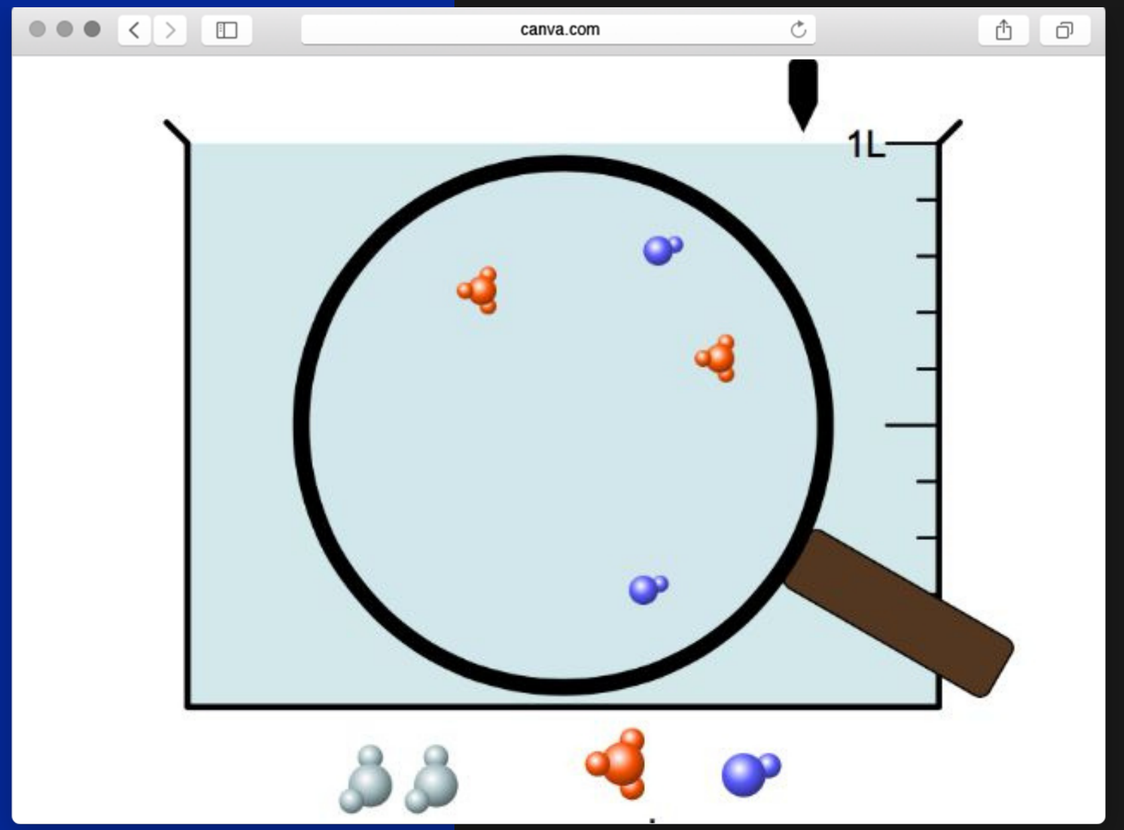
El método general consiste en:

1 Generar una lista de todas las soluciones potenciales del problema en una forma sistemática

2 Evaluar las soluciones potenciales una a una, descalificando las no factibles y manteniendo un registro de la mejor encontrada hasta el momento (en problemas de optimización)

3 Cuando la búsqueda finalice, regresar la mejor solución encontrada

Exploración exhaustiva



Exploración exhaustiva

La búsqueda exhaustiva es una técnica general de resolución de problemas.

Se realiza una búsqueda exhaustiva por el espacio de posibles soluciones hasta encontrar una que satisfaga los criterios exigidos. Por ello, suele resultar ineficiente.
La búsqueda se suele realizar recorriendo un árbol con el que se representan las posibles soluciones.

- Hay algunos métodos de recorrido del árbol:

- Recorrido en anchura.
- Backtracking.
- Branch and Bound (ramificación y acotación).

MÉTODO GENERAL

- La solución de un problema se puede expresar como una tupla (x1, x2, ..., xn), satisfaciendo unas restricciones P(x1, x2, ..., xn) y tal vez optimizando una cierta función objetivo.

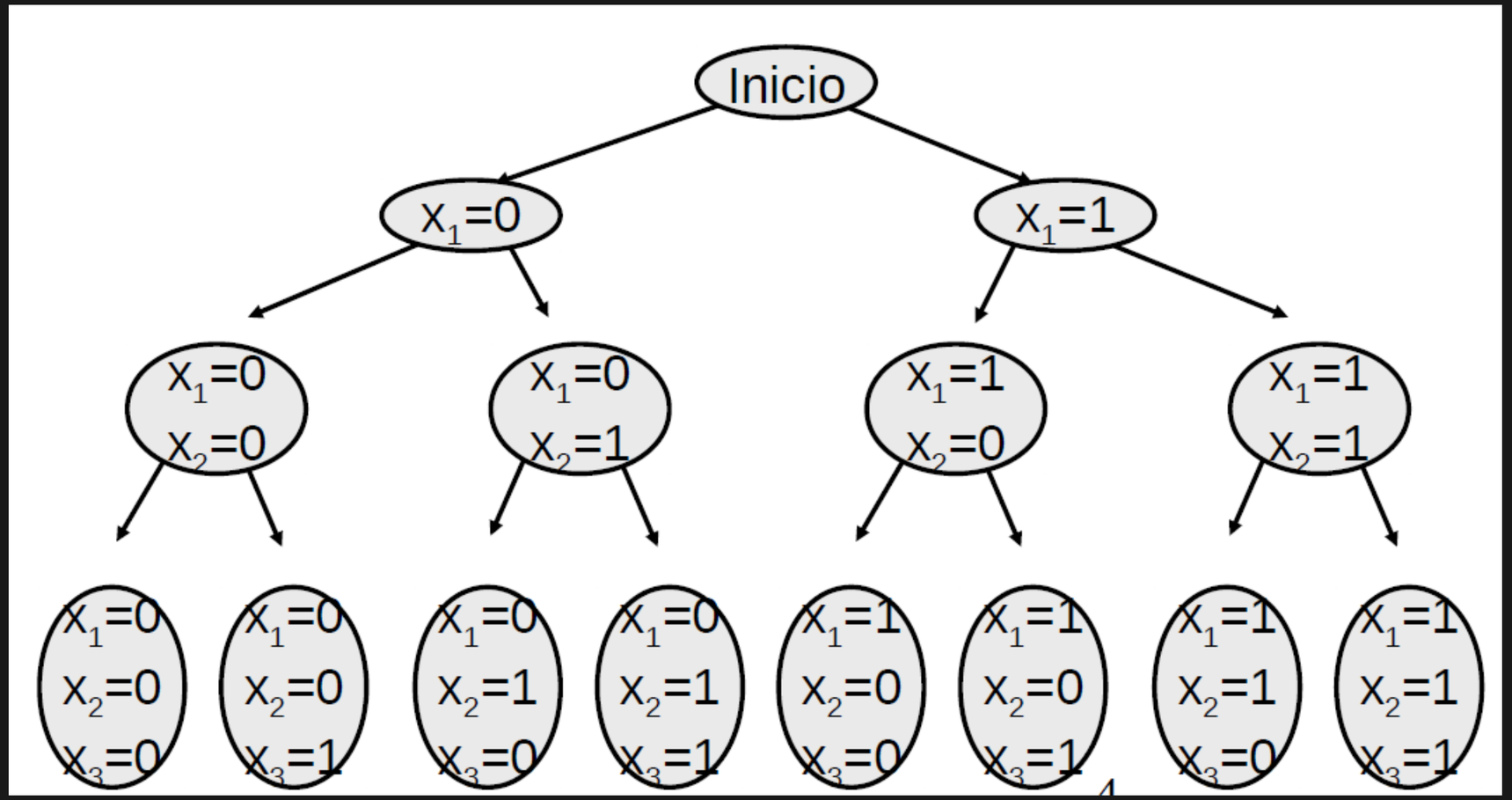
- En cada momento, el algoritmo se encontrará en un cierto nivel k, con una solución parcial (x1, ..., xk).

- Cada conjunto de posibles valores de la tupla representa un nodo del árbol de soluciones.

- Se sigue hasta que la solución parcial sea una solución completa del problema, o hasta que no queden más posibilidades por probar.

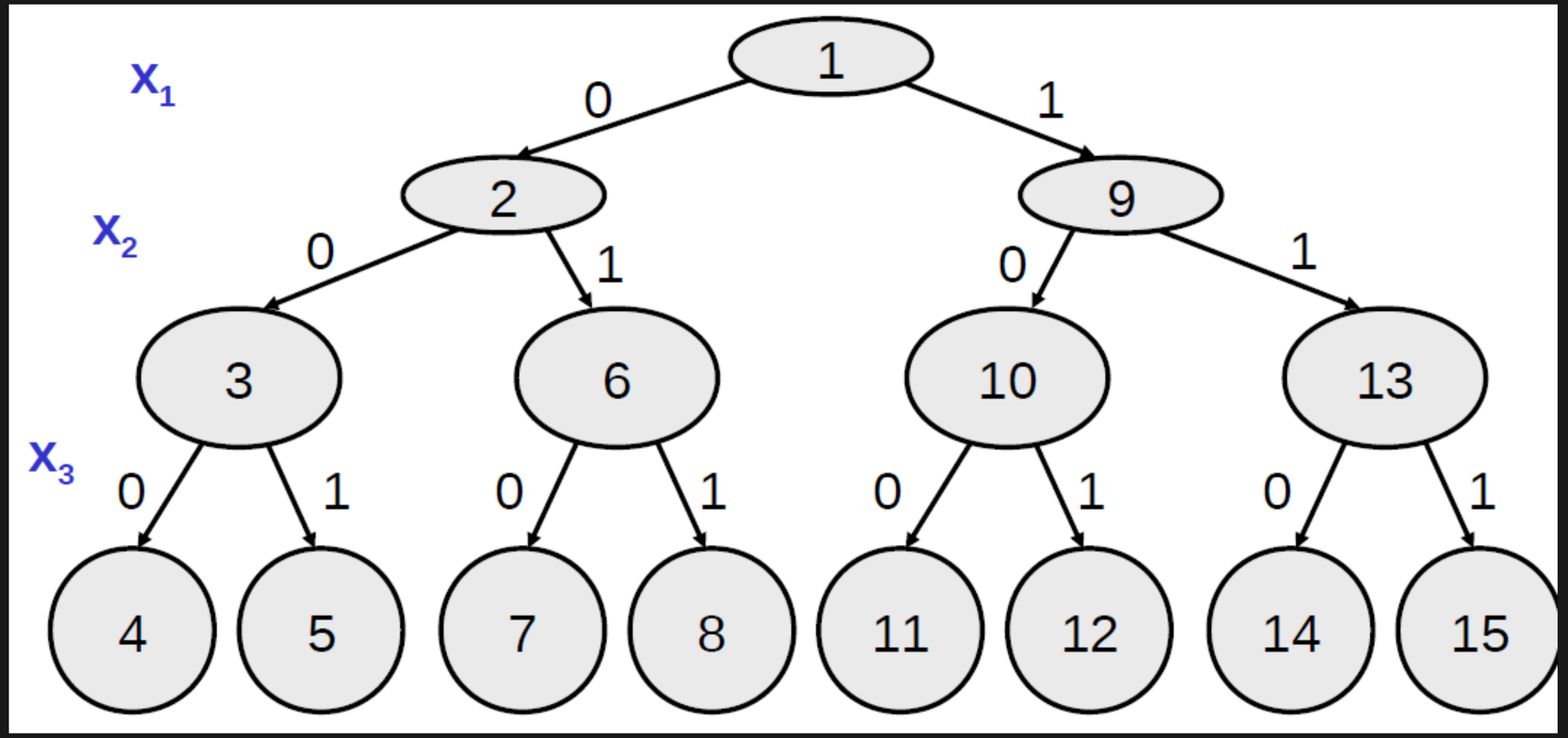
MÉTODO GENERAL

Se recorre un árbol de soluciones.
Sin embargo, este árbol implícito, no se almacena en ningún lugar.



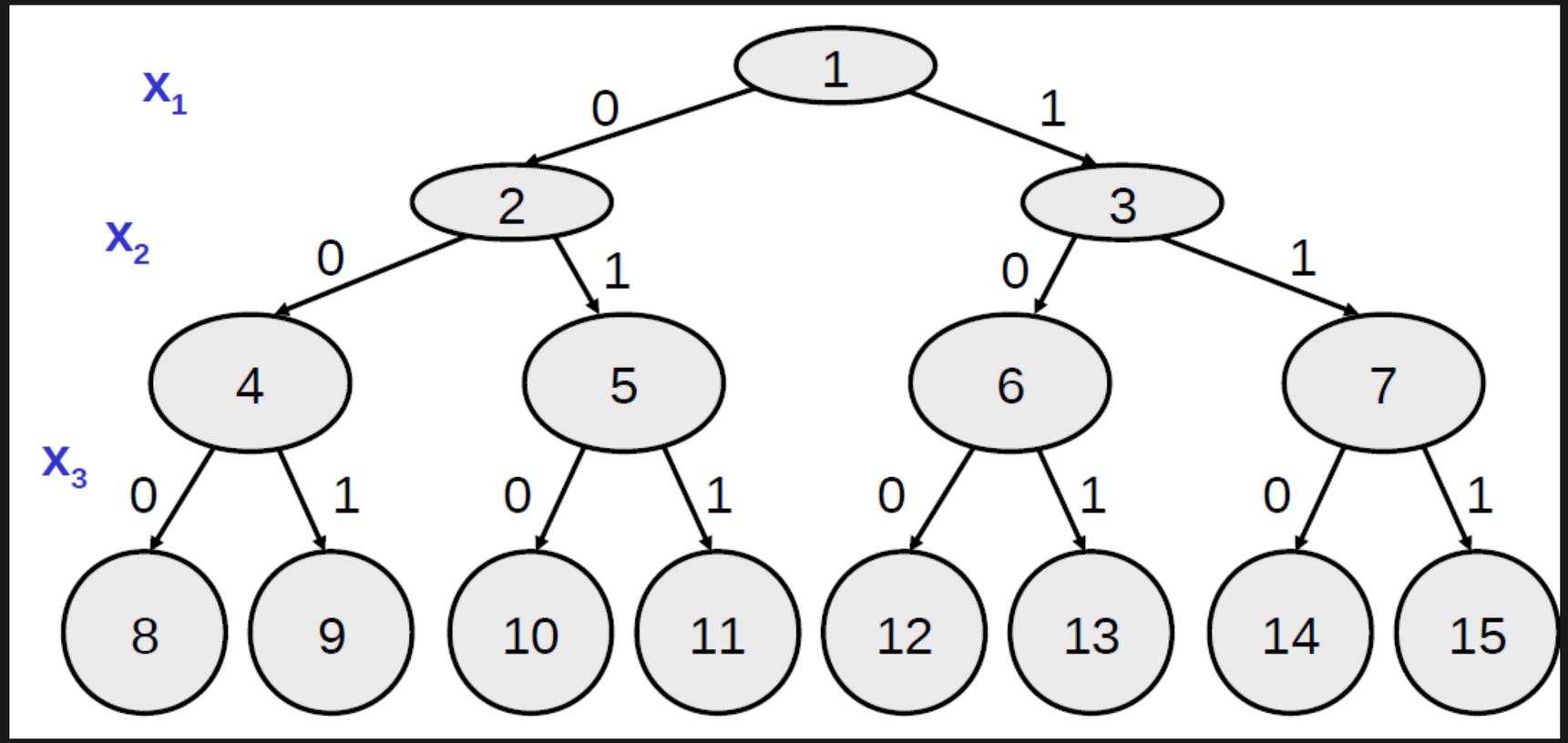
MÉTODO GENERAL

El recorrido se hace en un cierto orden. Por ejemplo, con backtracking se hace en profundidad:



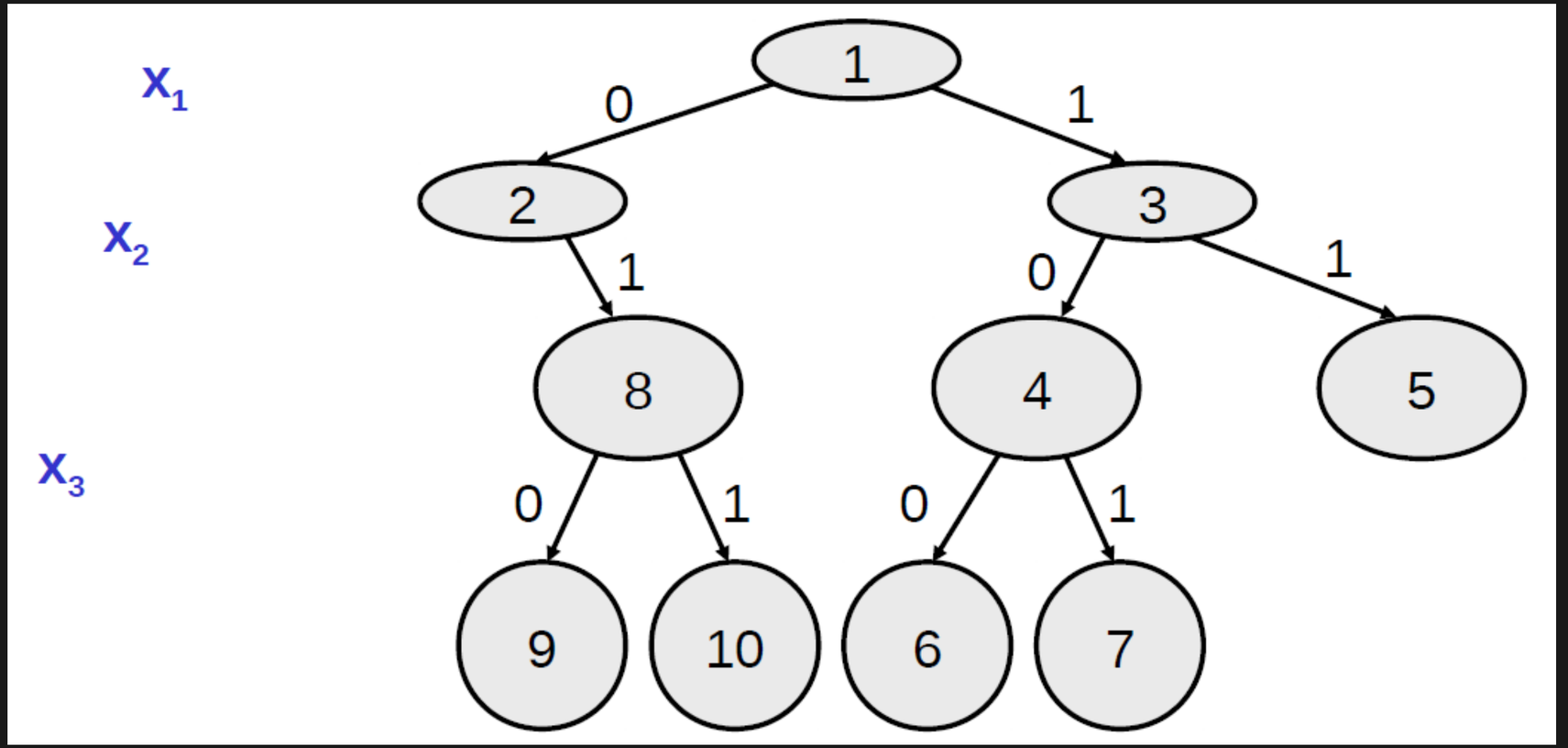
MÉTODO GENERAL

Y en anchura



MÉTODO GENERAL

Con Branch and Bound se guía la búsqueda por algún criterio, y se intenta eliminar nodos:



Ejemplo: El problema de la mochila

Observaciones
El problema de la mochila surge principalmente en los mecanismos de asignación de recursos. El nombre "Mochila" fue introducido por primera vez por Tobias Dantzig .
Espacio auxiliar: O(nw)
Complejidad del tiempo O(nw)

El problema : dado un conjunto de elementos en los que cada elemento contiene un peso y un valor, determine el número de cada uno para incluir en una colección de modo que el peso total sea menor o igual a un límite dado y el valor total sea lo más grande posible .

Pseudo código para problema de mochila

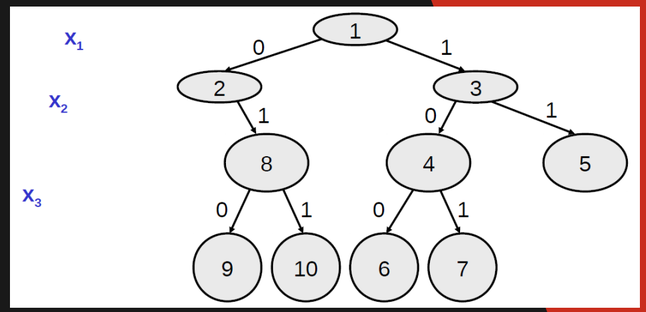
Dado:

1. Valores (array v)
2. Pesos (matriz w)
3. Número de elementos distintos (n)
4. Capacidad (W)

```
for j from 0 to W do:
    m[0, j] := 0
for i from 1 to n do:
    for j from 0 to W do:
        if w[i] > j then:
            m[i, j] := m[i-1, j]
        else:
            m[i, j] := max(m[i-1, j], m[i-1, j-w[i]] + v[i])
```

Branch and Bound (ramificación y acotación).

La técnica de Ramificación y poda se suele interpretar como un árbol de soluciones, donde cada rama nos lleva a una posible solución posterior a la actual. La característica de esta técnica con respecto a otras anteriores (y a la que debe su nombre) es que el algoritmo se encarga de detectar en qué ramificación las soluciones dadas ya no están siendo óptimas, para «podar» esa rama del árbol y no continuar malgastando recursos y procesos en casos que se alejan de la solución óptima.



Nuestra meta será encontrar el valor mínimo de una función f(x) (un ejemplo puede ser el coste de manufacturación de un determinado producto) donde fijamos x rangos sobre un determinado conjunto S de posibles soluciones. Un procedimiento de ramificación y poda requiere dos herramientas.

Pseudocódigo

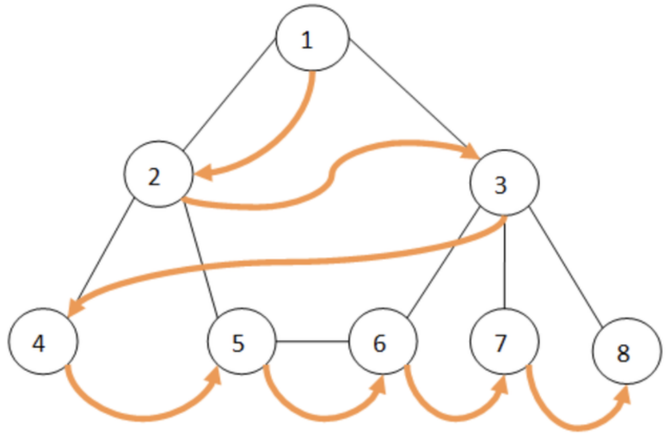
```
Funcion RyP {
 P = Hijos(x,k)
 while ( no vacio(P) )
    x(k) = extraer(P)
    if esFactible(x,k) y G(x,k) < optimo
        si esSolucion(x)
             Almacenar(x)
        else
             RyP(x,k+1)
```

Donde:

- **G(x)** es la función de estimación del algoritmo.
- **P** es la pila de posibles soluciones.
- **esFactible** es la función que considera si la propuesta es válida.
- **esSolución** es la función que comprueba si se satisface el objetivo.
- **óptimo** es el valor de la función a optimizar evaluado sobre la mejor solución encontrada hasta el momento.
- **NOTA**: Usamos un menor que (<) para los problemas de **minimización** y un mayor que (>) para problemas de **maximización**.

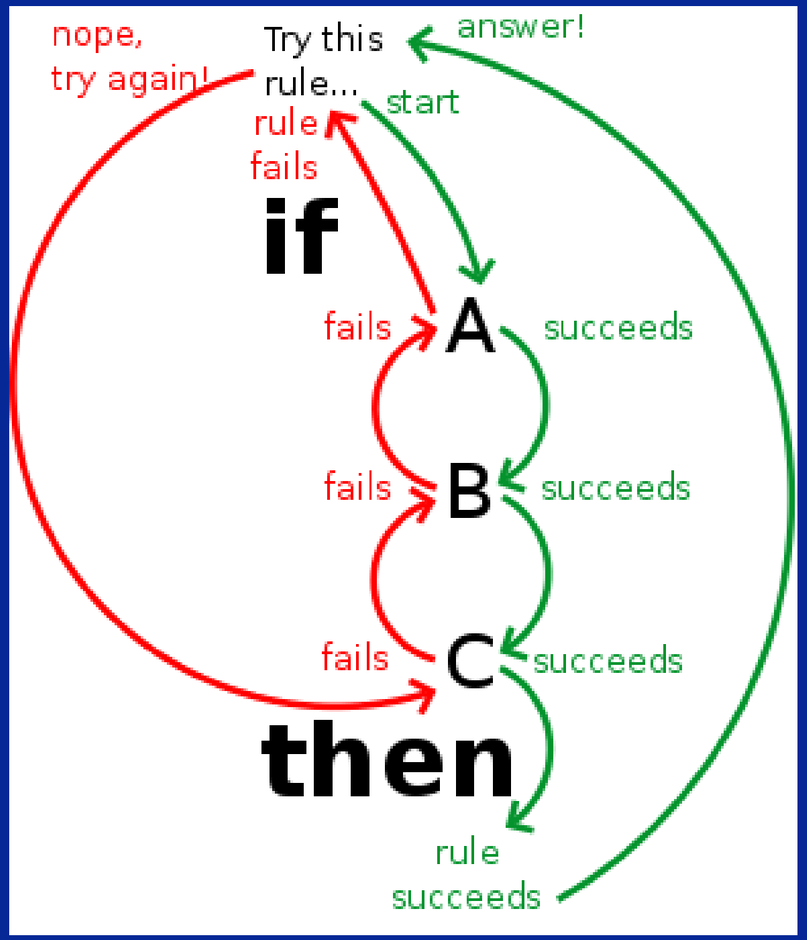
Búsqueda en Anchura

Una búsqueda en anchura (BFS) es un algoritmo de búsqueda para lo cual recorre los nodos de un grafo, comenzando en la raíz (eligiendo algún nodo como elemento raíz en el caso de un grafo), para luego explorar todos los vecinos de este nodo. A continuación, para cada uno de los vecinos se exploran sus respectivos vecinos adyacentes, y así hasta que se recorra todo el grafo. Cabe resaltar que si se encuentra el nodo antes de recorrer todos los nodos, concluye la búsqueda



La búsqueda por anchura se usa para aquellos algoritmos en donde resulta crítico elegir el mejor camino posible en cada momento del recorrido.

Programacion por vuelta atras



Introducción a la Vuelta Atrás

Los algoritmos de "Vuelta Atrás" (Backtracking) realizan una exploración sistemática de las posibles soluciones del problema

Vuelta Atrás suele utilizarse para resolver problemas complejos
• en los que la única forma de encontrar la solución es analizando todas las posibilidades
- ritmo de crecimiento exponencial
• constituye una forma sistemática de recorrer todo el espacio de soluciones

En general, podremos utilizar Vuelta Atrás para problemas:
• con solución representada por una n-tupla {x1,x2,...,xn}
• en los que cada una de las componentes xi de ese vector es elegida en cada etapa de entre un conjunto finito de valores (y1,y2,...,yv)

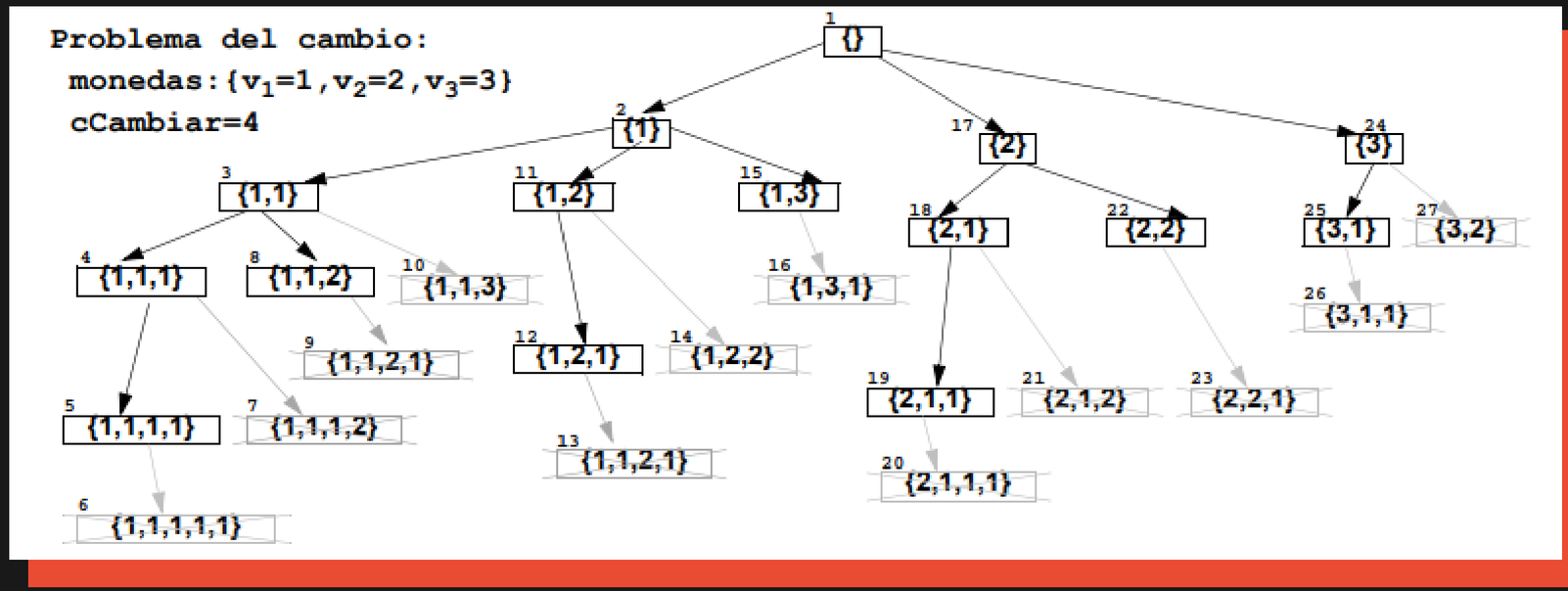
El algoritmo va obteniendo soluciones parciales:

• normalmente el espacio de soluciones parciales es un árbol

- árbol de búsqueda de las soluciones

• el algoritmo realiza un recorrido en profundidad del árbol

• cuando desde un nodo no se puede seguir buscando la solución se produce una vuelta atrás



Un recorrido tiene éxito si se llega a completar una solución. El objetivo del algoritmo puede ser encontrar una solución o encontrar todas las posibles soluciones al problema.

Los elementos principales del esquema de vuelta atrás son:

- IniciarExploraciónNivel(): recoge todas las opciones posibles en que se puede extender la solución k-prometedora.
- OpcionesPendientes(): comprueba que quedan opciones por explorar en el nivel.
- SoluciónCompleta(): comprueba que se haya completado una solución al problema.
- ProcesarSolución(): representa las operaciones que se quieran realizar con la solución, como imprimirla o devolverla al punto de llamada.
- Completable(): comprueba que la solución k-prometedora se puede extender con la opción elegida cumpliendo las restricciones del problema hasta llegar a completar una solución.

El esquema general de vuelta atrás que busca todas las soluciones puede formularse de la siguiente forma:

```
fun VueltaAtras (v: Secuencia, k: entero)
    {v es una secuencia k-prometedora}
    IniciarExploraciónNivel(k)
    mientras OpcionesPendientes(k) hacer
      extender v con siguiente opción
      si SoluciónCompleta(v) entonces
        ProcesarSolución( v)
      sino
        si Completable (v) entonces
          VueltaAtras(v, k+ 1)
        fsi
      fsi
   fmientras
ffun
```

Problema de las N reinas de ajedrez

Algunos conceptos :

- **Nodo vivo:** nodo del que todavía no se han generado todos sus hijos
- **Nodo prometedor:** nodo que no es solución pero desde el que todavía podría ser posible llegar a la solución

Dependiendo de si buscamos una solución cualquiera o la óptima

- el algoritmo se detiene una vez encontrada la primera solución
- o continúa buscando el resto de soluciones

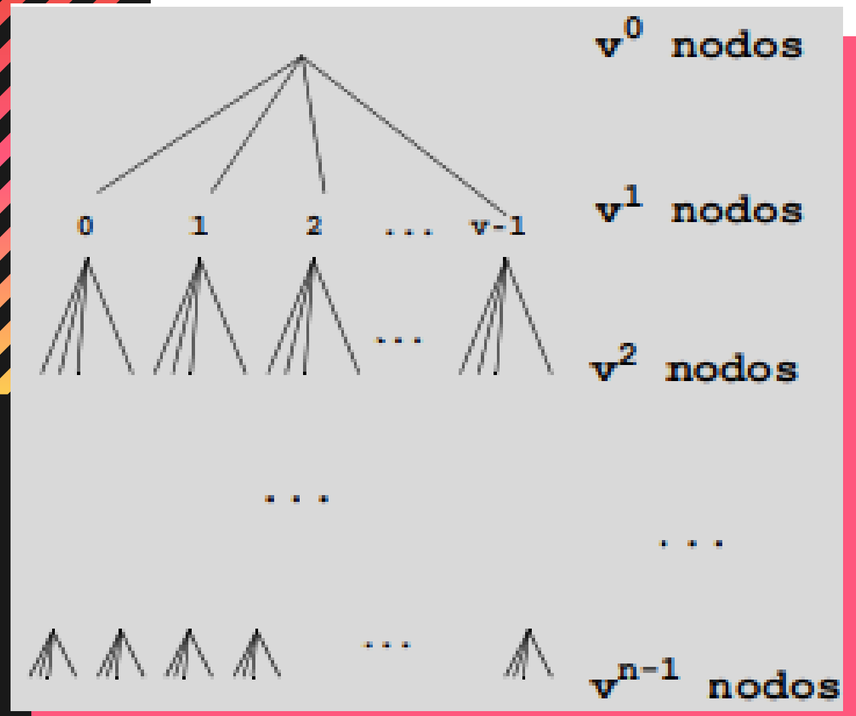
Estos algoritmos no crean ni gestionan el árbol explícitamente

- se crea implícitamente con las llamadas recursivas al algoritmo

Eficiencia de los algoritmos VA

La eficiencia de este tipo de algoritmos depende del número de nodos que sea necesario explorar para cada caso
- es imposible de calcular a priori de forma exacta
Pero es posible calcular una cota superior
• sea n la longitud máxima de la solución
y 0..v-1 el rango de valores para cada decisión tenemos que:

$$\text{nodos} = \sum_{i=0}^{n-1} v^i \cong v^n$$



luego su eficiencia temporal será O(vn)
- resultado muy pesimista en la mayoría de los casos
Tienen unos requisitos de memoria O(n)
- máxima "profundidad" de las llamadas recursivas

Nociones de complejidad de la exploración exhaustiva y vuelta atrás

Nociones de complejidad de progrmacion por vuelta atrás

En este esquema se muestran 3 elementos importantes de la VA, generación de descendientes de cada nodo como ***posible solución***, a cada uno de estos se le aplica el segundo elemento que es la ***prueba de fracaso,*** para finalmente aplicar el ultimo elemento la ***prueba de solución*** donde se comprueba si el nodo de posible solución en efecto es solución.

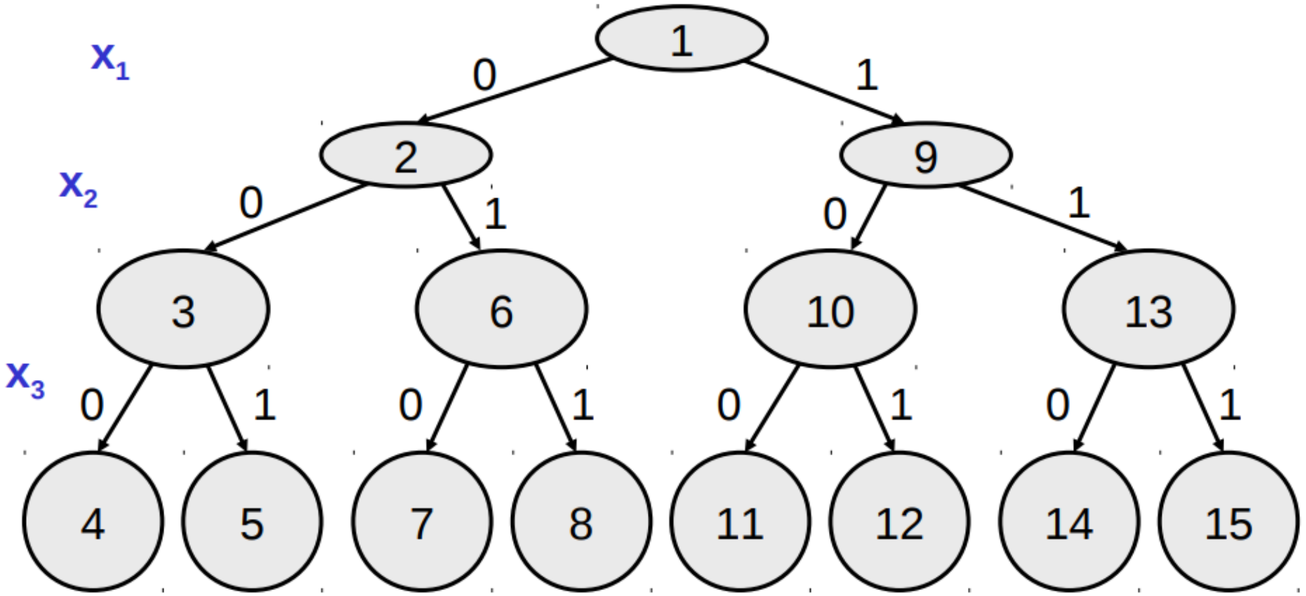
```
PROCEDURE VueltaAtras(etapa);
BEGIN
   IniciarOpciones;
   REPEAT
      SeleccionarNuevaOpcion;
      IF Aceptable THEN
         AnotarOpcion;
         IF SolucionIncompleta THEN
            VueltaAtras(etapa_siguiente);
            IF NOT exito THEN
               CancelarAnotacion
            END
         ELSE (* solucion completa *)
            exito:=TRUE
         END
      END
   UNTIL (exito) OR (UltimaOpcion)
END VueltaAtras;
```

Esquema general que poseen los algoritmos que siguen la técnica de Vuelta Atrás

vuelta atras

```
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)
hasta fin
```

Backtraking



Árbol por anchura