



Tema 9. Esquema Selección óptima.

Algorítmica y Complejidad

1

Introducción

Caracterización de los problemas

- **Tipo de problemas:**

- Problemas con **las mismas características** que los problemas de **backtracking**, pero con alguna restricción más:
 - Se debe encontrar la **mejor solución**.
 - Existe una función **objetivo** que devuelve la bondad de la solución hallada.
 - Hay que recorrer todo el árbol de búsqueda.

1. Introducción



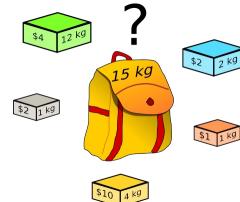
2

2

Introducción

Tipo de problemas. Ejemplos

- El problema de la mochila (Knapsack problem).
Dada una mochila incapaz de soportar más de un peso máximo y un conjunto de objetos, cada uno con un peso y valor específicos y de los que sólo existe una instancia, el problema consiste en meter en la mochila objetos, de tal forma que se maximice el valor de los objetos que contiene sin exceder el peso máximo que puede soportar la misma.



- El problema de las monedas.
Dado un sistema monetario con suficientes monedas de cada tipo y un importe a devolver, obtener el desglose de monedas para pagar el importe, de forma que el número de monedas empleado sea mínimo.



valores

1	2	5	10	20	50
---	---	---	----	----	----

Cantidad a devolver: 42

1. Introducción



3

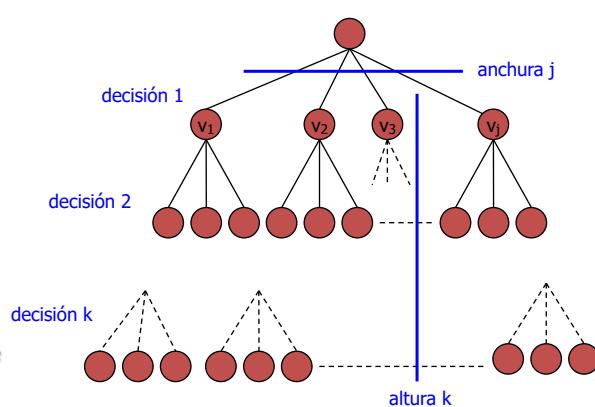
3

Introducción

Características del esquema

• Espacio de búsqueda:

- la altura del espacio: hay k estados por los que pasar para formar una solución.
- la anchura del espacio: cada estado tiene asociado un dominio formado por j decision/candidatos distintos.
- j^k hojas
- $n^{\text{ºnodos}} = \sum_{i=0}^k j^i \in O(j^k)$
- Puede que exista más de un espacio para el mismo problema. Por regla general se elige el más pequeño o el de generación menos costosa.



1. Introducción



4

4

Introducción

Características del esquema

- Modificaciones sobre el esquema general de backtracking para transformarlo en un esquema de selección óptima:
 1. Inclusión de **estructuras datos** para albergar la solución actual y **la mejor solución** alcanzada hasta el momento.
 2. El **procedimiento principal no parará** hasta haber **generado todas las posibles soluciones** (no se para ante la aparición de la primera solución).
 3. Siempre que se **realice** la operación de **anotar** se realizará la operación de **desanotar**.
 4. Cuando se alcanza una **solución** se comprobará **si es mejor que la mejor solución** alcanzada hasta el momento.
 5. Casi siempre se puede hacer **poda** sobre el árbol que representa el espacio de búsqueda, **evitando inspeccionar estados que no conducen a una solución mejor**.

1.Introducción



5

5

Introducción

Esquema general

```
void seleccionOptima (Estado estado, Solucion solucion, Valor actual
                      Solucion mejorSolucion, Valor mejor, ...){
    if (esSolucion(solucion)) {
        if (actual ES_MEJOR mejor)
            mejorSolucion = solucion;
            mejor = actual;
    }else if (hay_mas_candidatos) {
        for (candidatos){
            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                seleccionOptima(proximoEstado, solucion, actual,
                                mejorSolucion, mejor,...);
                desanotar_candidato();
            }
        }
    }
}
```

1.Introducción



6

6

Introducción

Esquema general

```

void seleccionOptima (Estado estado, Solucion solucion, Valor actual
                     Solucion mejorSolucion, Valor mejor, ...){
    if (esSolucion(solucion)) {
        if (actual ES_MEJOR mejor)
            mejorSolucion = solucion;
            mejor = actual;
    }else if (hay_mas_candidatos) {

        for (candidatos){

            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                seleccionOptima(proximoEstado, solucion, actual,
                                mejorSolucion, mejor,...);
                desanotar_candidato();
            }
        }
    }
}

```

Estructuras de datos

- **estado**: parámetros para mantener el estado actual de la búsqueda. Dependen del problema a resolver.
- **solucion**: parámetro para mantener la solución en curso.
- **actual**: bondad de la solución en curso.
- **mejorSolucion**: parámetro para mantener y finalmente devolver la mejor solución encontrada hasta el momento. **Java**: debe ser de tipo Objeto para que la modificación de sus propiedades se mantenga al devolver el control al estado anterior y así poder devolver la mejor solución encontrada.
- **mejor**: bondad de la mejor solución encontrada hasta el momento. **Java**: debe ser de tipo Objeto.

1.Introducción

1 2 3

7

Introducción

Esquema general

```

void seleccionOptima (Estado estado, Solucion solucion, Valor actual
                     Solucion mejorSolucion, Valor mejor, ...){
    if (esSolucion(solucion)) {
        if (actual ES_MEJOR mejor)
            mejorSolucion = solucion;
            mejor = actual;
    }else if (hay_mas_candidatos) {

        for (candidatos){

            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                seleccionOptima(proximoEstado, solucion, actual,
                                mejorSolucion, mejor,...);
                desanotar_candidato();
            }
        }
    }
}

```

Llamada inicial

Estado *estado* -> crear e inicializar variables que definen el estado
 Solucion *solucion* -> crear e inicializar variable para la solución en curso
 Valor *actual* -> bondad de la solución actual
 Solucion *mejorSolucion* -> crear e inicializar variable para la mejor solución
 (**Java**: debe ser de tipo Objeto)
 Valor *mejor* -> bondad de la mejor solución. **Inicializar con el peor valor**
 (**Java**: debe ser de tipo Objeto).
seleccionOptima(estado, solucion, actual, mejorSolucion, mejor, ...);
 if (mejor.getValor()==peor valor inicial) <>**no existe solución**>>
 else <>**solución encontrada y almacenada en mejorSolucion**>>

1.Introducción

1 2 3

8

Introducción

Esquema general

```

void seleccionOptima (Estado estado, Solucion solucion, Valor actual
                     Solucion mejorSolucion, Valor mejor, ...){
    if (esSolucion(solucion)) {
        if (actual ES_MEJOR mejor)
            mejorSolucion = solucion;
            mejor = actual;
    }else if (hay_mas_candidatos) {

        for (candidatos){

            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                seleccionOptima(proximoEstado, solucion, actual,
                                mejorSolucion, mejor,...);
                desanotar_candidato();
            }
        }
    }
}

```

Solución

- El estado actual es una solución: comprueba si la bondad de la solución actual es mejor que la bondad de la mejor solución y si es así actualiza *mejorSolucion* y *mejor* (mejor solución encontrada hasta el momento).
- Termina la ejecución del nivel actual y devuelve el control al nivel anterior (vuelta atrás).

1.Introducción

1 2 3

9

Introducción

Esquema general

```

void seleccionOptima (Estado estado, Solucion solucion, Valor actual
                     Solucion mejorSolucion, Valor mejor, ...){
    if (esSolucion(solucion)) {
        if (actual ES_MEJOR mejor)
            mejorSolucion = solucion;
            mejor = actual;
    }else if (hay_mas_candidatos) {

        for (candidatos){

            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                seleccionOptima(proximoEstado, solucion, actual,
                                mejorSolucion, mejor,...);
                desanotar_candidato();
            }
        }
    }
}

```

Quedan candidatos

- El nodo actual no es solución. Comprueba si en el nodo actual se pueden generar candidatos.

1.Introducción

1 2 3

10

Introducción

Esquema general

```

void seleccionOptima (Estado estado, Solucion solucion, Valor actual
                     Solucion mejorSolucion, Valor mejor, ...){
    if (esSolucion(solucion)) {
        if (actual ES_MEJOR mejor)
            mejorSolucion = solucion;
        mejor = actual;
    }else if (hay_mas_candidatos) {

        for (candidatos){

            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                seleccionOptima(proximoEstado, solucion, actual,
                                mejorSolucion, mejor,...);
                desanotar_candidato();
            }
        }
    }
}

```

Candidatos

- Bucle que itera por cada candidato. Al contrario que en backtracking, no sale del bucle al encontrar la primera solución: sigue inspeccionando candidatos hasta que no haya más.

1.Introducción

1 2 3

11

Introducción

Esquema general

```

void seleccionOptima (Estado estado, Solucion solucion, Valor actual
                     Solucion mejorSolucion, Valor mejor, ...){
    if (esSolucion(solucion)) {
        if (actual ES_MEJOR mejor)
            mejorSolucion = solucion;
        mejor = actual;
    }else if (hay_mas_candidatos) {

        for (candidatos){

            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                seleccionOptima(proximoEstado, solucion, actual,
                                mejorSolucion, mejor,...);
                desanotar_candidato();
            }
        }
    }
}

```

Aceptable

- Igual que en backtracking

1.Introducción

1 2 3

12

Introducción

Esquema general

```

void seleccionOptima (Estado estado, Solucion solucion, Valor actual
                     Solucion mejorSolucion, Valor mejor, ...){
    if (esSolucion(solucion)) {
        if (actual ES_MEJOR mejor)
            mejorSolucion = solucion;
        mejor = actual;
    }else if (hay_mas_candidatos) {

        for (candidatos){

            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                seleccionOptima(proximoEstado, solucion, actual,
                                mejorSolucion, mejor,...);
                desanotar_candidato();
            }
        }
    }
}

```

Anotar

- Además de actualizar las variables de *estado* e incorporar el candidato actual a la *solución* actual (tal y como se hace en el esquema de backtracking), **se debe actualizar la bondad de la solución actual**.

1.Introducción

1 2 3

13

Introducción

Esquema general

```

void seleccionOptima (Estado estado, Solucion solucion, Valor actual
                     Solucion mejorSolucion, Valor mejor, ...){
    if (esSolucion(solucion)) {
        if (actual ES_MEJOR mejor)
            mejorSolucion = solucion;
        mejor = actual;
    }else if (hay_mas_candidatos) {

        for (candidatos){

            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                seleccionOptima(proximoEstado, solucion, actual,
                                mejorSolucion, mejor,...);
                desanotar_candidato();
            }
        }
    }
}

```

Avance en el espacio de búsqueda

- Llamada recursiva (avanza al nodo inferior que surge del candidato recién anotado)

1.Introducción

1 2 3

14

Introducción

Esquema general

```

void seleccionOptima (Estado estado, Solucion solucion, Valor actual
                     Solucion mejorSolucion, Valor mejor, ...){
    if (esSolucion(solucion)) {
        if (actual ES_MEJOR mejor)
            mejorSolucion = solucion;
            mejor = actual;
    }else if (hay_mas_candidatos) {

        for (candidatos){
            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                if (actual ES_MEJOR mejor)
                    seleccionOptima(proximoEstado, solucion, actual,
                                    mejorSolucion, mejor,...);
                desanotar_candidato();
            }
        }
    }
}

```

Avance en el espacio de búsqueda

- Dependiendo del problema pueden hacerse **podas** del árbol de estados. Si la bondad de la solución actual (parcial) es igual o peor que la mejor encontrada hasta el momento y **el hecho de avanzar en el espacio de búsqueda no va a mejorarla**, se evita inspeccionar subárboles que con toda seguridad no contienen mejores soluciones que la mejor encontrada hasta el momento.

1.Introducción

1 2 3

15

Introducción

Esquema general

```

void seleccionOptima (Estado estado, Solucion solucion, Valor actual
                     Solucion mejorSolucion, Valor mejor, ...){
    if (esSolucion(solucion)) {
        if (actual ES_MEJOR mejor)
            mejorSolucion = solucion;
            mejor = actual;
    }else if (hay_mas_candidatos) {

        for (candidatos){
            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                seleccionOptima(proximoEstado, solucion, actual,
                                mejorSolucion, mejor,...);
                desanotar_candidato();
            }
        }
    }
}

```

Desanotar SIEMPRE a la vuelta

- A la vuelta de la llamada recursiva **SIEMPRE** se debe desanotar para dejar el estado actual tal y como estaba antes de anotar. De esta manera podemos volver a probar nuevos candidatos en el estado actual.
- actualizar las variables de *estado*, eliminar el candidato actual de la *solución* actual y actualizar la bondad de la solución *actual*.

1.Introducción

1 2 3

16

Introducción

Esquema general

```

void seleccionOptima (Estado estado, Solucion solucion, Valor actual
                     Solucion mejorSolucion, Valor mejor, ...){
    if (esSolucion(solucion)) {
        if (actual ES_MEJOR mejor)
            mejorSolucion = solucion;
            mejor = actual;
    }else if (hay_mas_candidatos) {

        for (candidatos){

            if (aceptable(candidato)){
                proximoEstado = anotar_candidato();
                seleccionOptima(proximoEstado, solucion, actual,
                                mejorSolucion, mejor,...);
                desanotar_candidato();
            }
        }
    }
}

```

La ejecución de este esquema hace que se recorra TODO el árbol de estados, buscando TODAS las posibles soluciones y manteniendo en todo momento en mejorSolucion la mejor de las soluciones que se haya encontrado hasta el momento.

The diagram illustrates a search tree where nodes are expanded from left to right. Solid lines represent valid transitions, while dashed lines represent invalid or pruned ones. The tree shows a path from the root node through several levels of children and grandchildren, with some branches being truncated by dashed lines.

1.Introducción

1 2 3

17

Introducción

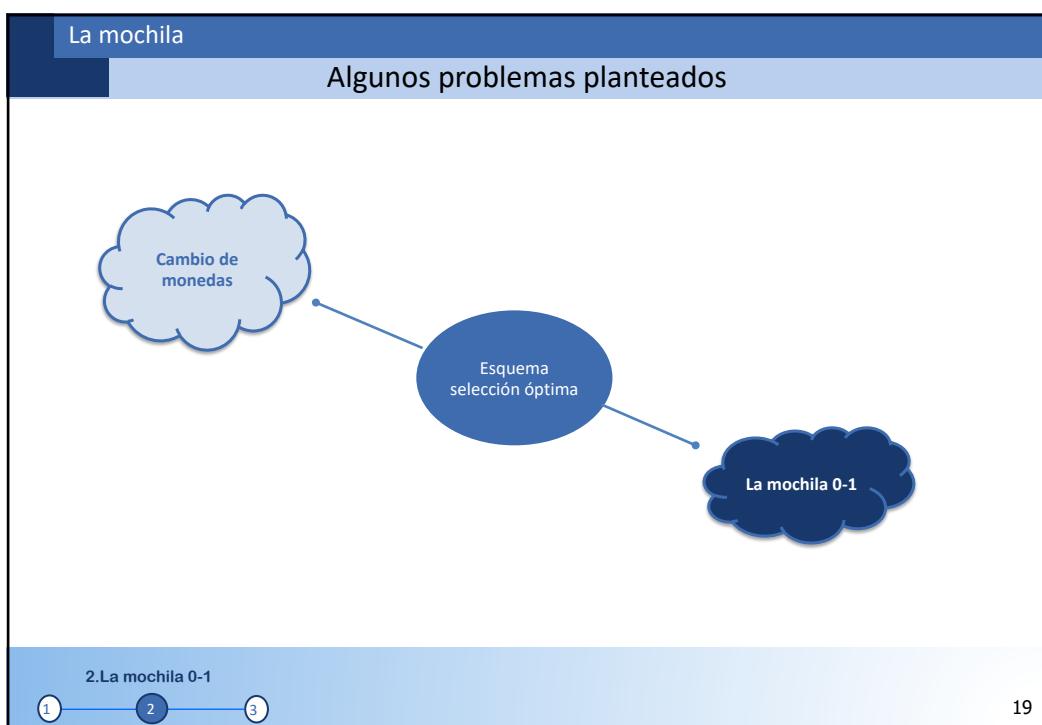
Algunos problemas planteados

The diagram features three light blue thought bubbles connected to a central dark blue circle labeled "Esquema selección óptima". The top-left bubble contains the text "Cambio de monedas". The bottom-right bubble contains the text "La mochila 0-1".

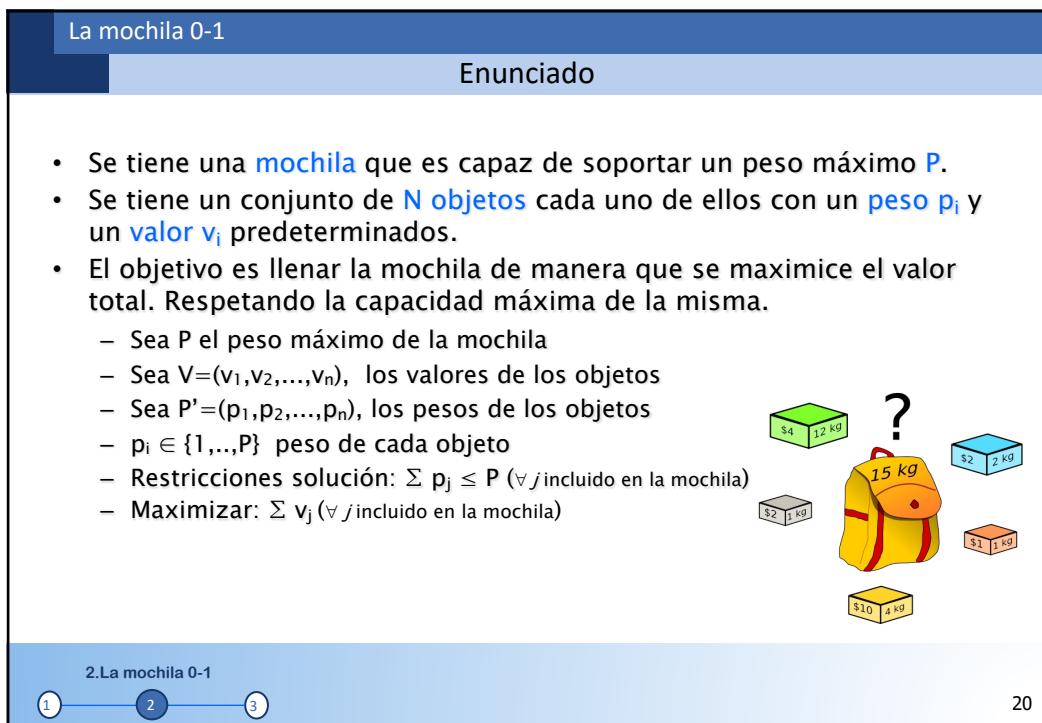
1.Introducción

1 2 3

18



19



20

La mochila 0-1

Espacio de búsqueda

Árbol de estados binario.
En cada nivel *i* decidir si el objeto *i* forma parte o no de la solución.

pesos	0	1	2
	2	3	7
valores	0	1	2
	35	21	37

Valor mochila: 0 37 21 58 35 72 56 93

nivel=0 (objeto pos. 0)
nivel=1 (objeto pos. 1)
nivel=2 (objeto pos. 2)

No desarrollar nodos con peso > MaxPeso

Mejor solución

2. La mochila 0-1

21

21

La mochila 0-1

Pasos

Nodo del árbol: trata el objeto de la posición *i* de los vectores *pesos* y *valores*.

Parámetros estado: *nivel*

Parámetros para solución: *solucion ->(x₀,..., x_{n-1})*, donde x_i = (0,1);
valorActual y *pesoActual* en la mochila.

Parámetros mejor solución: *mejorSolucion ->(x₀,..., x_{n-1})*, donde x_i = (0,1);
valorMejor en la mochila

Solución: si *nivel == pesos.length -> ¿valorActual > valorMejor? -> actualiza mejor*

Candidatos: no añadir/ añadir el objeto *i*-ésimo en la mochila

Aceptable: si candidato==añadir -> si añadir *pesos[i]* a la solución no supera el peso máximo de la mochila

Anotar: modificar *solucion[nivel]*, *valorActual* y *pesoActual*; *nivel = nivel + 1*;

Desanotar: *nivel = nivel - 1*; modificar *solucion[nivel]*, *valorActual* y *pesoActual*;

Llamada inicial: *mochilaOptima(pesos, valores, maxPeso, nivel, solucion, valorActual, pesoActual, mejorSolucion, valorMejor)*

2. La mochila 0-1

22

22

La mochila 0-1

Implementación

```

void mochilaOptima(int[] pesos, int[] valores, int maxPeso, int nivel, int[] solucion,
                    int valorActual, int pesoActual, int[] mejorSolucion, Entero valorMejor){
    if (nivel == pesos.length){
        if (valorActual > valorMejor.getValor()){
            valorMejor.setValor(valorActual);
            for (int i=0; i<solucion.length; i++) mejorSolucion[i]=solucion[i];
        }
    } else{
        for (int c=0; c<2; c++){
            if ((c==0) | (pesoActual + pesos[nivel]) <= maxPeso){
                solucion[nivel] = c;
                valorActual = valorActual + valores[nivel]*c;
                pesoActual = pesoActual + pesos[nivel]*c;
                nivel++;
                mochilaOptima(pesos, valores, maxPeso, nivel, solucion, valorActual,
                              pesoActual, mejorSolucion, valorMejor);
                nivel--;
                pesoActual = pesoActual - pesos[nivel]*c;
                valorActual = valorActual - valores[nivel]*c;
                solucion[nivel] = 0;
            }
        }
    }
}

```

2.La mochila 0-1



23

23

La mochila 0-1

Implementación

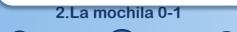
```

void mochilaOptima(int[] pesos, int[] valores, int maxPeso, int nivel, int[] solucion,
                    int valorActual, int pesoActual, int[] mejorSolucion, Entero valorMejor){
    if (nivel == pesos.length){
        if (valorActual > valorMejor.getValor()){
            valorMejor.setValor(valorActual);
            for (int i=0; i<solucion.length; i++) mejorSolucion[i]=solucion[i];
        }
    } else{
        for (int c=0; c<2; c++){
            if ((c==0) | (pesoActual + pesos[nivel]) <= maxPeso){
                solucion[nivel] = c;
                valorActual = valorActual + valores[nivel]*c;
                pesoActual = pesoActual + pesos[nivel]*c;
                nivel++;
                mochilaOptima(pesos, valores, maxPeso, nivel, solucion, valorActual,
                              pesoActual, mejorSolucion, valorMejor);
                nivel--;
                pesoActual = pesoActual - pesos[nivel]*c;
                valorActual = valorActual - valores[nivel]*c;
                solucion[nivel] = 0;
            }
        }
    }
}

public class Entero {
    private int valor;
    public Entero(int valor) {this.valor = valor;}
    public int getValor() {return this.valor;}
    public void setValor(int valor) {this.valor = valor;}
}

```

2.La mochila 0-1



24

24

La mochila 0-1

Implementación

```

void mochilaOptima(int[] pesos, int[] valores, int maxPeso, int nivel, int[] solucion,
                    int valorActual, int pesoActual, int[] mejorSolucion, Entero valorMejor){
    if (nivel == pesos.length){
        if (valorActual > valorMejor.getValor()){
            valorMejor.setValor(valorActual);
            for (int i=0; i<solucion.length; i++) mejorSolucion[i]=solucion[i];
        }
    } else{
        for (int c=0; c<2; c++){
            if ((c==0) | (pesoActual + pesos[nivel]) <= maxPeso){
                solucion[nivel] = c;
                valorActual = valorActual + valores[nivel]*c;
                pesoActual = pesoActual + pesos[nivel]*c;
                nivel++;
                mochilaOptima(pesos, valores, maxPeso, nivel, solucion, valorActual,
                              pesoActual, mejorSolucion, valorMejor);
                nivel--;
                pesoActual = pesoActual - pesos[nivel]*c;
                valorActual = valorActual - valores[nivel]*c;
                solucion[nivel] = 0;
            }
        }
    }
}

```

Solución

2.La mochila 0-1



25

25

La mochila 0-1

Implementación

```

void mochilaOptima(int[] pesos, int[] valores, int maxPeso, int nivel, int[] solucion,
                    int valorActual, int pesoActual, int[] mejorSolucion, Entero valorMejor){
    if (nivel == pesos.length){
        if (valorActual > valorMejor.getValor()){
            valorMejor.setValor(valorActual);
            for (int i=0; i<solucion.length; i++) mejorSolucion[i]=solucion[i];
        }
    } else{
        for (int c=0; c<2; c++){
            if ((c==0) | (pesoActual + pesos[nivel]) <= maxPeso){
                solucion[nivel] = c;
                valorActual = valorActual + valores[nivel]*c;
                pesoActual = pesoActual + pesos[nivel]*c;
                nivel++;
                mochilaOptima(pesos, valores, maxPeso, nivel, solucion, valorActual,
                              pesoActual, mejorSolucion, valorMejor);
                nivel--;
                pesoActual = pesoActual - pesos[nivel]*c;
                valorActual = valorActual - valores[nivel]*c;
                solucion[nivel] = 0;
            }
        }
    }
}

```

Quedan candidatos:
Sólo para nodos
intermedios (nunca
para nodos hoja)

2.La mochila 0-1



26

26

La mochila 0-1

Implementación

```

void mochilaOptima(int[] pesos, int[] valores, int maxPeso, int nivel, int[] solucion,
                    int valorActual, int pesoActual, int[] mejorSolucion, Entero valorMejor){
    if (nivel == pesos.length){
        if (valorActual > valorMejor.getValor()){
            valorMejor.setValor(valorActual);
            for (int i=0; i<solucion.length; i++) mejorSolucion[i]=solucion[i];
        }
    } else{
        for (int c=0; c<2; c++){
            if ((c==0) | (pesoActual + pesos[nivel]) <= maxPeso){
                solucion[nivel] = c;
                valorActual = valorActual + valores[nivel]*c;
                pesoActual = pesoActual + pesos[nivel]*c;
                nivel++;
                mochilaOptima(pesos, valores, maxPeso, nivel, solucion, valorActual,
                              pesoActual, mejorSolucion, valorMejor);
                nivel--;
                pesoActual = pesoActual - pesos[nivel]*c;
                valorActual = valorActual - valores[nivel]*c;
                solucion[nivel] = 0;
            }
        }
    }
}

```

Candidatos

2.La mochila 0-1

27

27

La mochila 0-1

Implementación

```

void mochilaOptima(int[] pesos, int[] valores, int maxPeso, int nivel, int[] solucion,
                    int valorActual, int pesoActual, int[] mejorSolucion, Entero valorMejor){
    if (nivel == pesos.length){
        if (valorActual > valorMejor.getValor()){
            valorMejor.setValor(valorActual);
            for (int i=0; i<solucion.length; i++) mejorSolucion[i]=solucion[i];
        }
    } else{
        for (int c=0; c<2; c++){
            if ((c==0) | (pesoActual + pesos[nivel]) <= maxPeso){           Candidato aceptable
                solucion[nivel] = c;
                valorActual = valorActual + valores[nivel]*c;
                pesoActual = pesoActual + pesos[nivel]*c;
                nivel++;
                mochilaOptima(pesos, valores, maxPeso, nivel, solucion, valorActual,
                              pesoActual, mejorSolucion, valorMejor);
                nivel--;
                pesoActual = pesoActual - pesos[nivel]*c;
                valorActual = valorActual - valores[nivel]*c;
                solucion[nivel] = 0;
            }
        }
    }
}

```

Candidato aceptable

2.La mochila 0-1

28

28

La mochila 0-1

Implementación

```

void mochilaOptima(int[] pesos, int[] valores, int maxPeso, int nivel, int[] solucion,
                    int valorActual, int pesoActual, int[] mejorSolucion, Entero valorMejor){
    if (nivel == pesos.length){
        if (valorActual > valorMejor.getValor()){
            valorMejor.setValor(valorActual);
            for (int i=0; i<solucion.length; i++) mejorSolucion[i]=solucion[i];
        }
    } else{
        for (int c=0; c<2; c++){
            if ((c==0) | (pesoActual + pesos[nivel]) <= maxPeso){
                solucion[nivel] = c;
                valorActual = valorActual + valores[nivel]*c;
                pesoActual = pesoActual + pesos[nivel]*c;
                nivel++;
                mochilaOptima(pesos, valores, maxPeso, nivel, solucion, valorActual,
                              pesoActual, mejorSolucion, valorMejor);
                nivel--;
                pesoActual = pesoActual - pesos[nivel]*c;
                valorActual = valorActual - valores[nivel]*c;
                solucion[nivel] = 0;
            }
        }
    }
}

```

anotar

2.La mochila 0-1

29

29

La mochila 0-1

Implementación

```

void mochilaOptima(int[] pesos, int[] valores, int maxPeso, int nivel, int[] solucion,
                    int valorActual, int pesoActual, int[] mejorSolucion, Entero valorMejor){
    if (nivel == pesos.length){
        if (valorActual > valorMejor.getValor()){
            valorMejor.setValor(valorActual);
            for (int i=0; i<solucion.length; i++) mejorSolucion[i]=solucion[i];
        }
    } else{
        for (int c=0; c<2; c++){
            if ((c==0) | (pesoActual + pesos[nivel]) <= maxPeso){
                solucion[nivel] = c;
                valorActual = valorActual + valores[nivel]*c;
                pesoActual = pesoActual + pesos[nivel]*c;
                nivel++;
                mochilaOptima(pesos, valores, maxPeso, nivel, solucion, valorActual,
                              pesoActual, mejorSolucion, valorMejor);
                nivel--;
                pesoActual = pesoActual - pesos[nivel]*c;
                valorActual = valorActual - valores[nivel]*c;
                solucion[nivel] = 0;
            }
        }
    }
}

```

Avance
en el
espacio
de
búsqueda

2.La mochila 0-1

30

30

La mochila 0-1

Implementación

```

void mochilaOptima(int[] pesos, int[] valores, int maxPeso, int nivel, int[] solucion,
                    int valorActual, int pesoActual, int[] mejorSolucion, Entero valorMejor){
    if (nivel == pesos.length){
        if (valorActual > valorMejor.getValor()){
            valorMejor.setValor(valorActual);
            for (int i=0; i<solucion.length; i++) mejorSolucion[i]=solucion[i];
        }
    } else{
        for (int c=0; c<2; c++){
            if ((c==0) | (pesoActual + pesos[nivel]) <= maxPeso){
                solucion[nivel] = c;
                valorActual = valorActual + valores[nivel]*c;
                pesoActual = pesoActual + pesos[nivel]*c;
                nivel++;
                mochilaOptima(pesos, valores, maxPeso, nivel, solucion, valorActual,
                              pesoActual, mejorSolucion, valorMejor);
                nivel--;
                pesoActual = pesoActual - pesos[nivel]*c;
                valorActual = valorActual - valores[nivel]*c;
                solucion[nivel] = 0;
            }
        }
    }
}

```

desanotar

2.La mochila 0-1

31

31

La mochila 0-1

Implementación

```

void mochilaOptima(int[] pesos, int[] valores, int maxPeso, int nivel, int[] solucion,
                    int valorActual, int pesoActual, int[] mejorSolucion, Entero valorMejor){
    if (nivel == pesos.length){
        if (valorActual > valorMejor.getValor()){
            valorMejor.setValor(valorActual);
            for (int i=0; i<solucion.length; i++) mejorSolucion[i]=solucion[i];
        }
    } else{
        for (int c=0; c<2; c++){
            if ((c==0) | (pesoActual + pesos[nivel]) <= maxPeso){
                solucion[nivel] = c;
                valorActual = valorActual + valores[nivel]*c;
                pesoActual = pesoActual + pesos[nivel]*c;
                nivel++;
                mochilaOptima(pesos, valores, maxPeso, 0, solucion,
                              valorActual, pesoActual, mejorSolucion, valorMejor);
                nivel--;
                pesoActual = pesoActual - pesos[nivel]*c;
                valorActual = valorActual - valores[nivel]*c;
                solucion[nivel] = 0;
            }
        }
    }
}

int[] pesos = {3, 7, 2};
int[] valores = {21, 37, 35};
int maxPeso = 10;

int[] solucion = new int[pesos.length];
int valorActual = 0;
int pesoActual = 0;

int[] mejorSolucion = new int[pesos.length];
Entero valorMejor = new Entero(0);

mochilaOptima(pesos, valores, maxPeso, 0, solucion,
              valorActual, pesoActual, mejorSolucion, valorMejor);

```

Llamada inicial

2.La mochila 0-1

32

32

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
2	3	7	

<i>valores</i>	0	1	2
35	21	37	



MaxPeso: 10

nivel = 0
 valorActual = 0
 pesoActual = 0
 solución

0	1	2
0	0	0
0	1	2

 mejorSolucion

0	0	0
0	0	0
0	0	0

 valorMejor = 0

1

2. La mochila 0-1

1 2 3

33

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
2	3	7	

<i>valores</i>	0	1	2
35	21	37	



MaxPeso: 10

Candidato: 0
 Aceptable. Anota

nivel = 1
 valorActual = 0
 pesoActual = 0
 solución

0	1	2
0	0	0
0	1	2

 mejorSolucion

0	0	0
0	0	0
0	0	0

 valorMejor = 0

1

2. La mochila 0-1

1 2 3

34

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
2	3	7	

<i>valores</i>	0	1	2
35	21	37	

MaxPeso: 10

nivel = 1
 valorActual = 0
 pesoActual = 0
 solución

0	1	2
0	0	0
0	1	2

 mejorSolucion

0	0	0
0	0	0
0	0	0

 valorMejor = 0

2.La mochila 0-1

1 ————— 2 ————— 3

35

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
2	3	7	

<i>valores</i>	0	1	2
35	21	37	

MaxPeso: 10

Candidato: 0
 Aceptable. Anota

nivel = 2
 valorActual = 0
 pesoActual = 0
 solución

0	1	2
0	0	0
0	1	2

 mejorSolucion

0	0	0
0	0	0
0	0	0

 valorMejor = 0

2.La mochila 0-1

1 ————— 2 ————— 3

36

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
2	3	7	

<i>valores</i>	0	1	2
35	21	37	

MaxPeso: 10

nivel = 2
valorActual = 0
pesoActual = 0
solucion [0 0 0]
mejorSolucion [0 0 0]
valorMejor = 0

2.La mochila 0-1

37

37

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
2	3	7	

<i>valores</i>	0	1	2
35	21	37	

MaxPeso: 10

Candidato: 0
Aceptable. Anota

nivel = 3
valorActual = 0
pesoActual = 0
solucion [0 0 0]
mejorSolucion [0 0 0]
valorMejor = 0

2.La mochila 0-1

38

38

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
2	3	7	

<i>valores</i>	0	1	2
35	21	37	

MaxPeso: 10

Diagram of a search tree for the knapsack problem:

```

graph TD
    N0(( )) --- N1((1))
    N0 --- N2((2))
    N0 --- N3((3))
    N2 --- N4((4))
    N4 --- N1
    N4 --- N2
    N4 --- N3
    N4 --- N0
  
```

Nodo hoja. NO mejora valorMejor

Variables en el cuadro:

- nivel = 3
- valorActual = 0
- pesoActual = 0
- solución

0	1	2
0	0	0

- mejorSolucion

0	1	2
0	0	0

- valorMejor = 0

2.La mochila 0-1

39

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
2	3	7	

<i>valores</i>	0	1	2
35	21	37	

MaxPeso: 10

Diagram of a search tree for the knapsack problem:

```

graph TD
    N0(( )) --- N1((1))
    N0 --- N2((2))
    N0 --- N3((3))
    N3 --- N4(( ))
    N4 --- N1
    N4 --- N2
    N4 --- N3
    N4 --- N0
  
```

Vuelta de recursividad

Variables en el cuadro:

- nivel = 3
- valorActual = 0
- pesoActual = 0
- solución

0	1	2
0	0	0

- mejorSolucion

0	1	2
0	0	0

- valorMejor = 0

2.La mochila 0-1

40

La mochila 0-1

Espacio de búsqueda

pesos

0	1	2
2	3	7


 MaxPeso: 10

valores

0	1	2
35	21	37

Desanota

nivel = 2
 valorActual = 0
 pesoActual = 0
 solución

0	0	0
0	1	2

 mejorSolucion

0	0	0
---	---	---

 valorMejor = 0

2.La mochila 0-1

41

La mochila 0-1

Espacio de búsqueda

pesos

0	1	2
2	3	7


 MaxPeso: 10

valores

0	1	2
35	21	37

Candidato: 1
Aceptable. Anota

nivel = 3
 valorActual = 37
 pesoActual = 7
 solución

0	0	1
0	1	2

 mejorSolucion

0	0	0
---	---	---

 valorMejor = 0

2.La mochila 0-1

42

La mochila 0-1

Espacio de búsqueda

pesos	0	1	2
	2	3	7

valores	0	1	2
	35	21	37

MaxPeso: 10

Diagram of a search tree for the knapsack problem:

```

graph TD
    N0(( )) --- N1((1))
    N0 --- N2((2))
    N2 --- N3((3))
    N2 --- N4((4))
    N3 --- N5((5))
    N3 --- N6(( ))
    N4 --- N7(( ))
    N5 --- N8(( ))
    N5 --- N9(( ))
  
```

Labels on edges: 0, 0, 0, 1, 0, 0.

Annotations for node 5 (highlighted in red):

- Nodo hoja. Mejora valorMejor
- nivel = 3
- valorActual = 37
- pesoActual = 7
- solución

0	1	2
0	0	1
- mejorSolucion

0	1	2
0	0	0
- valorMejor = 0

2.La mochila 0-1

43

43

La mochila 0-1

Espacio de búsqueda

pesos	0	1	2
	2	3	7

valores	0	1	2
	35	21	37

MaxPeso: 10

Diagram of a search tree for the knapsack problem:

```

graph TD
    N0(( )) --- N1((1))
    N0 --- N2((2))
    N2 --- N3((3))
    N2 --- N4((4))
    N3 --- N5((5))
    N3 --- N6(( ))
    N4 --- N7(( ))
    N5 --- N8(( ))
    N5 --- N9(( ))
  
```

Labels on edges: 0, 0, 0, 1, 0, 0.

Annotations for node 5 (highlighted in red):

- Nodo hoja. Mejora valorMejor
- Actualiza mejorSolucion y valorMejor
- nivel = 3
- valorActual = 37
- pesoActual = 7
- solución

0	1	2
0	0	1
- mejorSolucion

0	1	2
0	0	1
- valorMejor = 37

2.La mochila 0-1

44

44

La mochila 0-1

Espacio de búsqueda

pesos

0	1	2
2	3	7


 MaxPeso: 10

valores

0	1	2
35	21	37

Vuelta de recursividad

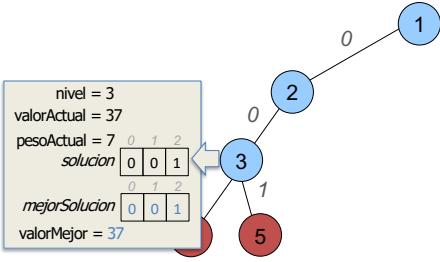
nivel = 3
 valorActual = 37
 pesoActual = 7
 solución

0	1	2
0	0	1

 mejorSolucion

0	0	1
---	---	---

 valorMejor = 37



2.La mochila 0-1

45

45

La mochila 0-1

Espacio de búsqueda

pesos

0	1	2
2	3	7


 MaxPeso: 10

valores

0	1	2
35	21	37

Desanota

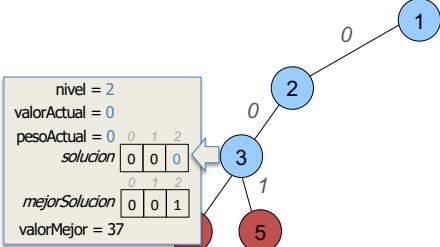
nivel = 2
 valorActual = 0
 pesoActual = 0
 solución

0	1	2
0	0	0

 mejorSolucion

0	0	1
---	---	---

 valorMejor = 37



2.La mochila 0-1

46

46

La mochila 0-1

Espacio de búsqueda

pesos	0	1	2
	2	3	7

valores	0	1	2
	35	21	37

MaxPeso: 10

Vuelta de recursividad

```

graph TD
    1((1)) -- 0 --> 2((2))
    1 -- 1 --> 4((4))
    1 -- 1 --> 5((5))
    2 -- 0 --> 3((3))
    2 -- 1 --> 4
    2 -- 1 --> 5
  
```

nivel = 2
valorActual = 0
pesoActual = 0
solucion

0	1	2
0	0	0

mejorSolucion

0	1	2
0	0	1

valorMejor = 37

2.La mochila 0-1

47

La mochila 0-1

Espacio de búsqueda

pesos	0	1	2
	2	3	7

valores	0	1	2
	35	21	37

MaxPeso: 10

Desanota

```

graph TD
    1((1)) -- 0 --> 2((2))
    1 -- 1 --> 4((4))
    1 -- 1 --> 5((5))
    2 -- 0 --> 3((3))
    2 -- 1 --> 4
    2 -- 1 --> 5
  
```

nivel = 1
valorActual = 0
pesoActual = 0
solucion

0	1	2
0	0	0

mejorSolucion

0	1	2
0	0	1

valorMejor = 37

2.La mochila 0-1

48

La mochila 0-1

Espacio de búsqueda

pesos	0	1	2
	2	3	7

valores	0	1	2
	35	21	37

MaxPeso: 10

Candidato: 1
Aceptable. Anota

nivel = 2
valorActual = 21
pesoActual = 3
solucion [0 1 0]
mejorSolucion [0 0 1]
valorMejor = 37

2.La mochila 0-1

49

La mochila 0-1

Espacio de búsqueda

pesos	0	1	2
	2	3	7

valores	0	1	2
	35	21	37

MaxPeso: 10

nivel = 2
valorActual = 21
pesoActual = 3
solucion [0 1 0]
mejorSolucion [0 0 1]
valorMejor = 37

2.La mochila 0-1

50

La mochila 0-1

Espacio de búsqueda

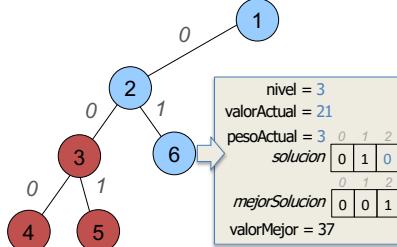
pesos	0	1	2
	2	3	7

valores	0	1	2
	35	21	37


MaxPeso: 10

nivel = 3
 valorActual = 21
 pesoActual = 3
 solución [0 1 0]
 [0 1 2]
 mejorSolución [0 0 1]
 valorMejor = 37

Candidato: 0
Aceptable. Anota



```

graph TD
    1((1)) -- 0 --> 2((2))
    1 -- 1 --> 6((6))
    2 -- 0 --> 3((3))
    2 -- 1 --> 6
    3 -- 0 --> 4((4))
    3 -- 1 --> 5((5))
    
```

2. La mochila 0-1

51

La mochila 0-1

Espacio de búsqueda

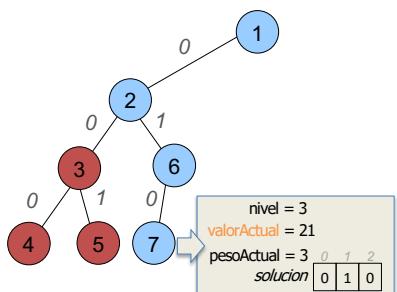
pesos	0	1	2
	2	3	7

valores	0	1	2
	35	21	37


MaxPeso: 10

nivel = 3
 valorActual = 21
 pesoActual = 3
 solución [0 1 0]
 [0 1 2]
 mejorSolución [0 0 1]
 valorMejor = 37

Nodo hoja. NO mejora valorMejor



```

graph TD
    1((1)) -- 0 --> 2((2))
    1 -- 1 --> 6((6))
    2 -- 0 --> 3((3))
    2 -- 1 --> 6
    3 -- 0 --> 4((4))
    3 -- 1 --> 5((5))
    6 -- 0 --> 7((7))
    
```

2. La mochila 0-1

52

La mochila 0-1

Espacio de búsqueda

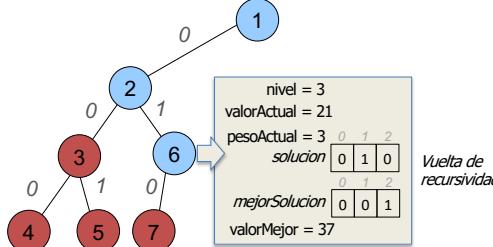
pesos

0	1	2
2	3	7


 MaxPeso: 10

valores

0	1	2
35	21	37



```

graph TD
    1((1)) -- 0 --> 2((2))
    1 -- 0 --> 6((6))
    2 -- 0 --> 3((3))
    2 -- 1 --> 6
    3 -- 0 --> 4((4))
    3 -- 1 --> 5((5))
    3 -- 0 --> 7((7))
  
```

2.La mochila 0-1

53

La mochila 0-1

Espacio de búsqueda

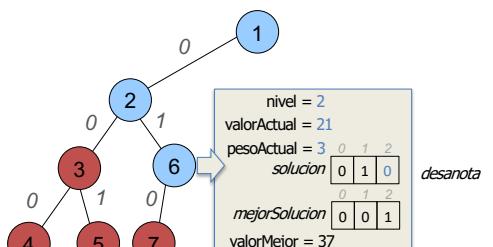
pesos

0	1	2
2	3	7


 MaxPeso: 10

valores

0	1	2
35	21	37



```

graph TD
    1((1)) -- 0 --> 2((2))
    1 -- 0 --> 6((6))
    2 -- 0 --> 3((3))
    2 -- 1 --> 6
    3 -- 0 --> 4((4))
    3 -- 1 --> 5((5))
    3 -- 0 --> 7((7))
  
```

2.La mochila 0-1

54

La mochila 0-1

Espacio de búsqueda

pesos	0	1	2
	2	3	7

valores	0	1	2
	35	21	37

MaxPeso: 10

Árbol de búsqueda:

```

graph TD
    N1((1)) --- N2((2))
    N2 --- N3((3))
    N2 --- N6((6))
    N3 --- N4((4))
    N3 --- N5((5))
    N3 --- N7((7))
    N6 --- N8((8))
    
```

Nodos visitados:

- Nodo 1: nivel = 3, valorActual = 58, pesoActual = 10, solución = [0 1 1], mejorSolución = [0 0 1], valorMejor = 37. Se marca como "Candidato: 1 Aceptable. Anota".
- Nodos 2, 3, 4, 5, 6, 7: No están marcados.
- Nodo 8: No está marcado.

2. La mochila 0-1

55

La mochila 0-1

Espacio de búsqueda

pesos	0	1	2
	2	3	7

valores	0	1	2
	35	21	37

MaxPeso: 10

Árbol de búsqueda:

```

graph TD
    N1((1)) --- N2((2))
    N2 --- N3((3))
    N2 --- N6((6))
    N3 --- N4((4))
    N3 --- N5((5))
    N3 --- N7((7))
    N6 --- N8((8))
    
```

Nodos visitados:

- Nodo 1: nivel = 3, valorActual = 58, pesoActual = 10, solución = [0 1 1], mejorSolución = [0 0 1], valorMejor = 37. Se marca como "Candidato: 1 Aceptable. Anota".
- Nodos 2, 3, 4, 5, 6, 7: No están marcados.
- Nodo 8: Se marca como "Nodo hoja. Mejora valorMejor".

2. La mochila 0-1

56

La mochila 0-1

Espacio de búsqueda

pesos	0	1	2
2	3	7	

valores	0	1	2
35	21	37	

MaxPeso: 10

Arbol de búsqueda:

```

graph TD
    N1((1)) --- N2_0((2))
    N2_0 --- N3_0((3))
    N2_0 --- N6((6))
    N3_0 --- N4((4))
    N3_0 --- N5((5))
    N6 --- N7((7))
    N6 --- N8((8))
    
```

Nodo hoja. Mejora valorMejor
Actualiza mejorSolucion y valorMejor

nivel = 3
valorActual = 58
pesoActual = 10
solucion [0 | 1 | 1]
mejorSolucion [0 | 1 | 1]
valorMejor = 58

2.La mochila 0-1

57

La mochila 0-1

Espacio de búsqueda

pesos	0	1	2
2	3	7	

valores	0	1	2
35	21	37	

MaxPeso: 10

Arbol de búsqueda:

```

graph TD
    N1((1)) --- N2_0((2))
    N2_0 --- N3_0((3))
    N2_0 --- N6((6))
    N3_0 --- N4((4))
    N3_0 --- N5((5))
    N6 --- N7((7))
    N6 --- N8((8))
    
```

Vuelta de recursividad

nivel = 3
valorActual = 58
pesoActual = 10
solucion [0 | 1 | 1]
mejorSolucion [0 | 1 | 1]
valorMejor = 58

2.La mochila 0-1

58

La mochila 0-1

Espacio de búsqueda

pesos	0	1	2
	2	3	7

valores	0	1	2
	35	21	37


MaxPeso: 10

nivel = 2
 valorActual = 21
 pesoActual = 3
 solución [0 | 1 | 0]
 mejorSolución [0 | 1 | 1]
 valorMejor = 58

Desanota

```

graph TD
    0((0)) -- 0 --> 1((1))
    0 -- 1 --> 2((2))
    2 -- 0 --> 3((3))
    2 -- 1 --> 6((6))
    3 -- 0 --> 4((4))
    3 -- 1 --> 5((5))
    6 -- 0 --> 7((7))
    6 -- 1 --> 8((8))
    style 0 fill:#0070C0,stroke:#0070C0
    style 1 fill:#0070C0,stroke:#0070C0
    style 2 fill:#0070C0,stroke:#0070C0
    style 3 fill:#C0392B,stroke:#C0392B
    style 4 fill:#C0392B,stroke:#C0392B
    style 5 fill:#C0392B,stroke:#C0392B
    style 6 fill:#0070C0,stroke:#0070C0
    style 7 fill:#C0392B,stroke:#C0392B
    style 8 fill:#C0392B,stroke:#C0392B
    
```

2.La mochila 0-1

59

La mochila 0-1

Espacio de búsqueda

pesos	0	1	2
	2	3	7

valores	0	1	2
	35	21	37


MaxPeso: 10

nivel = 2
 valorActual = 21
 pesoActual = 3
 solución [0 | 1 | 0]
 mejorSolución [0 | 1 | 1]
 valorMejor = 58

Vuelta de recursividad

```

graph TD
    0((0)) -- 0 --> 1((1))
    0 -- 1 --> 2((2))
    2 -- 0 --> 3((3))
    2 -- 1 --> 6((6))
    3 -- 0 --> 4((4))
    3 -- 1 --> 5((5))
    6 -- 0 --> 7((7))
    6 -- 1 --> 8((8))
    style 0 fill:#0070C0,stroke:#0070C0
    style 1 fill:#0070C0,stroke:#0070C0
    style 2 fill:#0070C0,stroke:#0070C0
    style 3 fill:#C0392B,stroke:#C0392B
    style 4 fill:#C0392B,stroke:#C0392B
    style 5 fill:#C0392B,stroke:#C0392B
    style 6 fill:#0070C0,stroke:#0070C0
    style 7 fill:#C0392B,stroke:#C0392B
    style 8 fill:#C0392B,stroke:#C0392B
    
```

2.La mochila 0-1

60

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
2	3	7	

<i>valores</i>	0	1	2
35	21	37	


MaxPeso: 10

Desanota

```

nivel = 1
valorActual = 0
pesoActual = 0
solucion [0 0 0]
[0 1 2]
mejorSolucion [0 1 1]
valorMejor = 58

```

2.La mochila 0-1

61

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
2	3	7	

<i>valores</i>	0	1	2
35	21	37	


MaxPeso: 10

Vuelta de recursividad

```

nivel = 1
valorActual = 0
pesoActual = 0
solucion [0 0 0]
[0 1 2]
mejorSolucion [0 1 1]
valorMejor = 58

```

2.La mochila 0-1

62

La mochila 0-1

Espacio de búsqueda

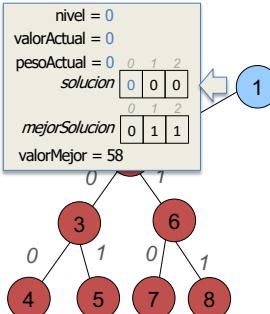
pesos 0 1 2
 pesos 2 3 7

valores 0 1 2
 valores 35 21 37

 MaxPeso: 10

Desanota

nivel = 0
 valorActual = 0
 pesoActual = 0 0 1 2
 solución 0 0 0
 mejorSolucion 0 1 1
 valorMejor = 58



2.La mochila 0-1

63

63

La mochila 0-1

Espacio de búsqueda

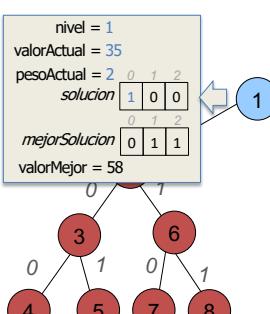
pesos 0 1 2
 pesos 2 3 7

valores 0 1 2
 valores 35 21 37

 MaxPeso: 10

Candidato: 1
 Aceptable. Anota

nivel = 1
 valorActual = 35
 pesoActual = 2 0 1 2
 solución 1 0 0
 mejorSolucion 0 1 1
 valorMejor = 58



2.La mochila 0-1

64

64

La mochila 0-1

Espacio de búsqueda

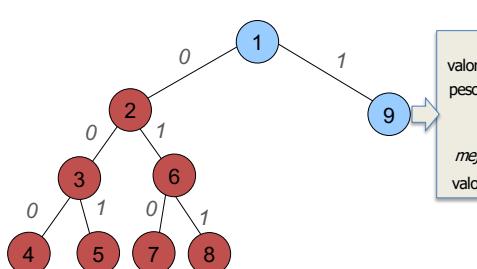
pesos

0	1	2
2	3	7

 valores

0	1	2
35	21	37

 MaxPeso: 10



nivel = 1
 valorActual = 35
 pesoActual = 2

0	1	2
1	0	0

 solución

0	1	2
0	1	1

 mejorSolucion

0	1	1
0	1	2

 valorMejor = 58

2.La mochila 0-1

1 → 2 → 3

65

La mochila 0-1

Espacio de búsqueda

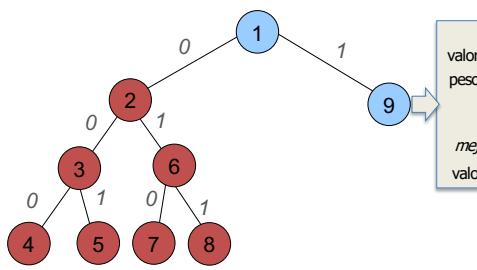
pesos

0	1	2
2	3	7

 valores

0	1	2
35	21	37

 MaxPeso: 10



nivel = 2
 valorActual = 35
 pesoActual = 2

0	1	2
1	0	0

 solución

0	1	2
0	1	1

 mejorSolucion

0	1	1
0	1	2

 valorMejor = 58

Candidato: 0
 Aceptable. Anota

2.La mochila 0-1

1 → 2 → 3

66

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
2	3	7	

<i>valores</i>	0	1	2
35	21	37	


MaxPeso: 10

Arbol de búsqueda:

```

graph TD
    N1((1)) --- N2_0((2))
    N1 --- N10((10))
    N2_0 --- N3_0((3))
    N2_0 --- N6((6))
    N3_0 --- N4((4))
    N3_0 --- N5((5))
    N6 --- N7((7))
    N6 --- N8((8))
    
```

Nodo actual: 10

Datos del nodo actual:

- nivel = 2
- valorActual = 35
- pesoActual = 2
- solución:

0	1	2
1	0	0
- mejorSolucion:

0	1	2
0	1	1
- valorMejor = 58

2.La mochila 0-1

67

67

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
2	3	7	

<i>valores</i>	0	1	2
35	21	37	


MaxPeso: 10

Arbol de búsqueda:

```

graph TD
    N1((1)) --- N2_0((2))
    N1 --- N10((10))
    N2_0 --- N3_0((3))
    N2_0 --- N6((6))
    N3_0 --- N4((4))
    N3_0 --- N5((5))
    N6 --- N7((7))
    N6 --- N8((8))
    
```

Nodo actual: 10

Datos del nodo actual:

- nivel = 3
- valorActual = 35
- pesoActual = 2
- solución:

0	1	2
1	0	0
- mejorSolucion:

0	1	2
0	1	1
- valorMejor = 58

Candidato: 0
Aceptable. Anota

2.La mochila 0-1

68

68

La mochila 0-1

Espacio de búsqueda

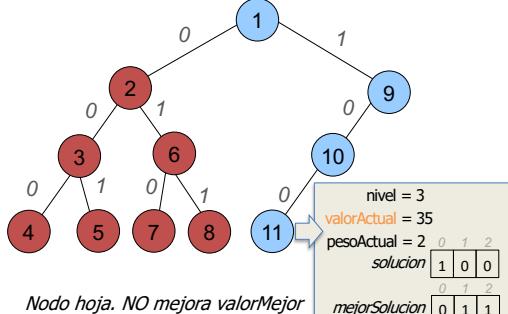
pesos

0	1	2
2	3	7

valores

0	1	2
35	21	37

 MaxPeso: 10



Nodo hoja. NO mejora valorMejor

2.La mochila 0-1

69

La mochila 0-1

Espacio de búsqueda

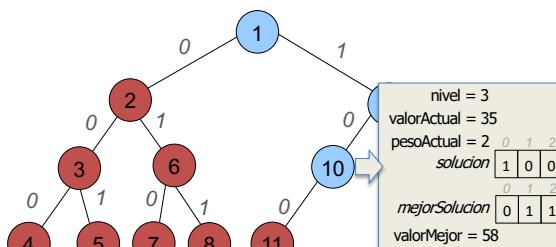
pesos

0	1	2
2	3	7

valores

0	1	2
35	21	37

 MaxPeso: 10



Vuelta de recursividad

2.La mochila 0-1

70

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
	2	3	7

<i>valores</i>	0	1	2
	35	21	37


MaxPeso: 10

Arbol de búsqueda:

```

graph TD
    N1((1)) --- N2_0((2))
    N1 --- N10((10))
    N2_0 --- N3_0((3))
    N2_0 --- N6((6))
    N3_0 --- N4((4))
    N3_0 --- N5((5))
    N6 --- N7((7))
    N6 --- N8((8))
    N10 --- N11((11))
    
```

Nodos visitados (azules): 1, 10

Nodos explorados (rojos): 2, 3, 6, 4, 5, 7, 8, 11

Nodo actual (azul): 10

Datos del nodo actual:

- nivel = 2
- valorActual = 35
- pesoActual = 2 0 1 2
- solución:

1	0	0
0	1	2
- mejorSolucion:

0	1	1
0	1	2
- valorMejor = 58

Desanota

2.La mochila 0-1

1 —> 2 —> 3

71

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
	2	3	7

<i>valores</i>	0	1	2
	35	21	37


MaxPeso: 10

Arbol de búsqueda:

```

graph TD
    N1((1)) --- N2_0((2))
    N1 --- N10((10))
    N2_0 --- N3_0((3))
    N2_0 --- N6((6))
    N3_0 --- N4((4))
    N3_0 --- N5((5))
    N6 --- N7((7))
    N6 --- N8((8))
    N10 --- N11((11))
    
```

Nodos visitados (azules): 1, 10

Nodos explorados (rojos): 2, 3, 6, 4, 5, 7, 8, 11

Nodo actual (azul): 10

Datos del nodo actual:

- nivel = 3
- valorActual = 72
- pesoActual = 9 0 1 2
- solución:

1	0	1
0	1	2
- mejorSolucion:

0	1	1
0	1	2
- valorMejor = 58

Candidato: 1
Aceptable. Anota

2.La mochila 0-1

1 —> 2 —> 3

72

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
	2	3	7

<i>valores</i>	0	1	2
	35	21	37

MaxPeso: 10

Arbol de búsqueda:

```

graph TD
    N1((1)) --- N2((2))
    N1 --- N9((9))
    N2 --- N3((3))
    N2 --- N6((6))
    N3 --- N4((4))
    N3 --- N5((5))
    N6 --- N7((7))
    N6 --- N8((8))
    N9 --- N10((10))
    N9 --- N12((12))
    N10 --- N11((11))
    N10 --- N12
  
```

Nodo hoja. Mejora valorMejor

nivel = 3
 valorActual = 72
 pesoActual = 9
 solución

0	1	2
1	0	1
0	1	2

 mejorSolucion

0	1	2
1	0	1
0	1	2

 valorMejor = 58

2.La mochila 0-1

1 —> 2 —> 3

73

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
	2	3	7

<i>valores</i>	0	1	2
	35	21	37

MaxPeso: 10

Arbol de búsqueda:

```

graph TD
    N1((1)) --- N2((2))
    N1 --- N9((9))
    N2 --- N3((3))
    N2 --- N6((6))
    N3 --- N4((4))
    N3 --- N5((5))
    N6 --- N7((7))
    N6 --- N8((8))
    N9 --- N10((10))
    N9 --- N12((12))
    N10 --- N11((11))
    N10 --- N12
  
```

Nodo hoja. Mejora valorMejor
 Actualiza mejorSolucion y valorMejor

nivel = 3
 valorActual = 72
 pesoActual = 9
 solución

0	1	2
1	0	1
0	1	2

 mejorSolucion

1	0	1
1	0	1
0	1	2

 valorMejor = 72

2.La mochila 0-1

1 —> 2 —> 3

74

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
	2	3	7

<i>valores</i>	0	1	2
	35	21	37


MaxPeso: 10

Arbol de búsqueda:

```

graph TD
    N1((1)) --- N2_0((2))
    N1 --- N10_0((10))
    N2_0 --- N3_0((3))
    N2_0 --- N6_0((6))
    N10_0 --- N11_0((11))
    N10_0 --- N12_0((12))
    N3_0 --- N4_0((4))
    N3_0 --- N5_0((5))
    N6_0 --- N7_0((7))
    N6_0 --- N8_0((8))
    
```

Nodos visitados:

- Nodo 1 (raíz)
- Nodo 2 (hijo de 1)
- Nodo 3 (hijo de 2)
- Nodo 6 (hijo de 2)
- Nodo 10 (hijo de 1)
- Nodo 11 (hijo de 10)
- Nodo 12 (hijo de 10)
- Nodos 4, 5, 7, 8 (hijos de 3 y 6 respectivamente)

Datos en el cuadro:

nivel = 3
valorActual = 72
pesoActual = 9
solucion

0	1	2
1	0	1

mejorSolucion

1	0	1
---	---	---

valorMejor = 72

Vuelta de recursividad

2.La mochila 0-1

75

75

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
	2	3	7

<i>valores</i>	0	1	2
	35	21	37


MaxPeso: 10

Arbol de búsqueda:

```

graph TD
    N1((1)) --- N2_0((2))
    N1 --- N10_0((10))
    N2_0 --- N3_0((3))
    N2_0 --- N6_0((6))
    N10_0 --- N11_0((11))
    N10_0 --- N12_0((12))
    N3_0 --- N4_0((4))
    N3_0 --- N5_0((5))
    N6_0 --- N7_0((7))
    N6_0 --- N8_0((8))
    
```

Nodos visitados:

- Nodo 1 (raíz)
- Nodo 2 (hijo de 1)
- Nodo 3 (hijo de 2)
- Nodo 6 (hijo de 2)
- Nodo 10 (hijo de 1)
- Nodo 11 (hijo de 10)
- Nodo 12 (hijo de 10)
- Nodos 4, 5, 7, 8 (hijos de 3 y 6 respectivamente)

Datos en el cuadro:

nivel = 2
valorActual = 35
pesoActual = 2
solucion

0	1	2
1	0	0

mejorSolucion

1	0	1
---	---	---

valorMejor = 72

Desanota

2.La mochila 0-1

76

76

La mochila 0-1

Espacio de búsqueda

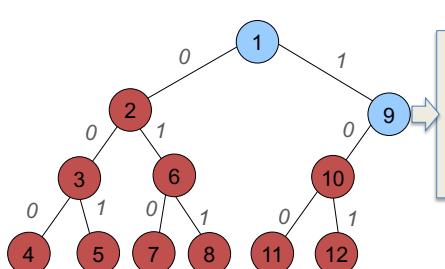
pesos

0	1	2
2	3	7

 valores

0	1	2
35	21	37


 MaxPeso: 10



nivel = 2
 valorActual = 35
 pesoActual = 2

0	1	2
1	0	0

 solucion

0	1	2
1	0	1

 mejorSolucion

0	1	2
1	0	1

 valorMejor = 72

Vuelta de recursividad

2. La mochila 0-1

1 → 2 → 3

77

La mochila 0-1

Espacio de búsqueda

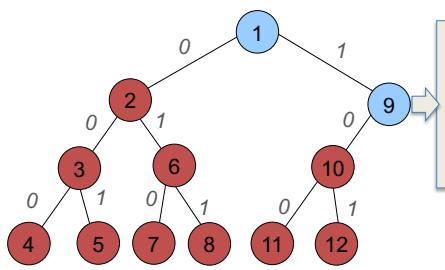
pesos

0	1	2
2	3	7

 valores

0	1	2
35	21	37


 MaxPeso: 10



nivel = 1
 valorActual = 35
 pesoActual = 2

0	1	2
1	0	0

 solucion

0	1	2
1	0	1

 mejorSolucion

0	1	2
1	0	1

 valorMejor = 72

Desanota

2. La mochila 0-1

1 → 2 → 3

78

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
	2	3	7

<i>valores</i>	0	1	2
	35	21	37



MaxPeso: 10

Arbol de búsqueda:

```

graph TD
    N1((1)) --- N2((2))
    N1 --- N9((9))
    N2 --- N3((3))
    N2 --- N6((6))
    N3 --- N4((4))
    N3 --- N5((5))
    N6 --- N7((7))
    N6 --- N8((8))
    N9 --- N10((10))
    N9 --- N11((11))
    N10 --- N11((11))
    N10 --- N12((12))
    
```

Nodo actual: 9

Datos del nodo actual:

- nivel = 2
- valorActual = 56
- pesoActual = 5
- solución:

1	1	0
0	1	2
- mejorSolucion:

1	0	1
0	1	2
- valorMejor = 72

Candidato: 1
Aceptable. Anota

2. La mochila 0-1

79

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
	2	3	7

<i>valores</i>	0	1	2
	35	21	37



MaxPeso: 10

Arbol de búsqueda:

```

graph TD
    N1((1)) --- N2((2))
    N1 --- N9((9))
    N2 --- N3((3))
    N2 --- N6((6))
    N3 --- N4((4))
    N3 --- N5((5))
    N6 --- N7((7))
    N6 --- N8((8))
    N9 --- N10((10))
    N9 --- N13((13))
    N10 --- N11((11))
    N10 --- N12((12))
    
```

Nodo actual: 13

Datos del nodo actual:

- nivel = 2
- valorActual = 56
- pesoActual = 5
- solución:

1	1	0
0	1	2
- mejorSolucion:

1	0	1
0	1	2
- valorMejor = 72

2. La mochila 0-1

80

La mochila 0-1

Espacio de búsqueda

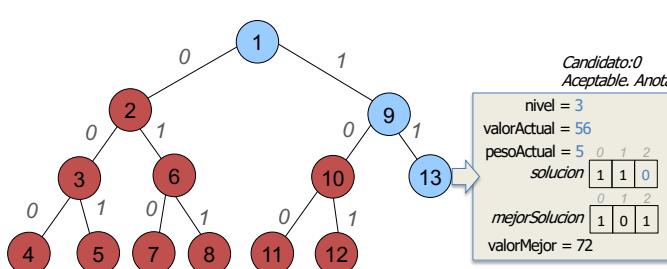
pesos

0	1	2
2	3	7

valores

0	1	2
35	21	37

 MaxPeso: 10



2.La mochila 0-1

81

La mochila 0-1

Espacio de búsqueda

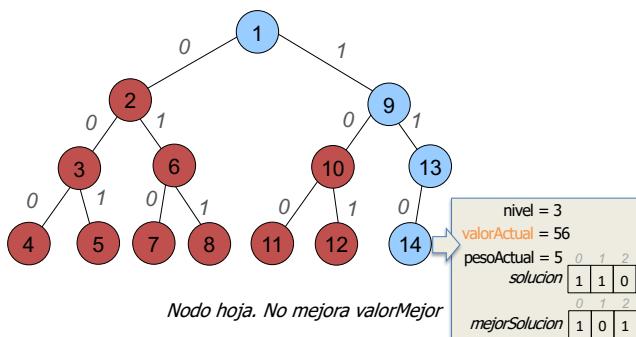
pesos

0	1	2
2	3	7

valores

0	1	2
35	21	37

 MaxPeso: 10



2.La mochila 0-1

82

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
2	3	7	

<i>valores</i>	0	1	2
35	21	37	


MaxPeso: 10

Arbol de búsqueda:

```

graph TD
    N1((1)) --- N2_0((2))
    N1 --- N9_0((9))
    N2_0 --- N3_0((3))
    N2_0 --- N6_0((6))
    N9_0 --- N10_0((10))
    N9_0 --- N13((13))
    N3_0 --- N4_0((4))
    N3_0 --- N5_0((5))
    N6_0 --- N7_0((7))
    N6_0 --- N8_0((8))
    N10_0 --- N11_0((11))
    N10_0 --- N12_0((12))
    N10_0 --- N14_0((14))
    
```

Vuelta de recursividad:

nivel = 3			
valorActual = 56			
pesoActual = 5	0	1	2
solución	1	1	0
mejorSolucion	1	0	1
valorMejor = 72			

2.La mochila 0-1

83

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
2	3	7	

<i>valores</i>	0	1	2
35	21	37	


MaxPeso: 10

Arbol de búsqueda:

```

graph TD
    N1((1)) --- N2_0((2))
    N1 --- N9_0((9))
    N2_0 --- N3_0((3))
    N2_0 --- N6_0((6))
    N9_0 --- N10_0((10))
    N9_0 --- N13((13))
    N3_0 --- N4_0((4))
    N3_0 --- N5_0((5))
    N6_0 --- N7_0((7))
    N6_0 --- N8_0((8))
    N10_0 --- N11_0((11))
    N10_0 --- N12_0((12))
    N10_0 --- N14_0((14))
    
```

Desanota:

nivel = 2			
valorActual = 56			
pesoActual = 5	0	1	2
solución	1	1	0
mejorSolucion	1	0	1
valorMejor = 72			

2.La mochila 0-1

84

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
	2	3	7

<i>valores</i>	0	1	2
	35	21	37



MaxPeso: 10

Árbol de búsqueda:

```

graph TD
    N1((1)) --- N2_0((2))
    N1 --- N9_0((9))
    N2_0 --- N3_0((3))
    N2_0 --- N6_0((6))
    N9_0 --- N10_0((10))
    N9_0 --- N13_0((13))
    N3_0 --- N4_0((4))
    N3_0 --- N5_0((5))
    N6_0 --- N7_0((7))
    N6_0 --- N8_0((8))
    N10_0 --- N11_0((11))
    N10_0 --- N12_0((12))
    N13_0 --- N14_0((14))
    
```

Nodos visitados: 1, 2, 3, 6, 10, 13

Nodo actual: 13 (No aceptable)

Datos de búsqueda:

- Candidato: 1 NO Aceptable (5+7>10)
- nivel = 2
- valorActual = 56
- pesoActual = 5
- solución:

0	1	2
1	1	0
- mejorSolucion:

1	0	1
0	1	2
- valorMejor = 72

2.La mochila 0-1

85

85

La mochila 0-1

Espacio de búsqueda

<i>pesos</i>	0	1	2
	2	3	7

<i>valores</i>	0	1	2
	35	21	37



MaxPeso: 10

Árbol de búsqueda:

```

graph TD
    N1((1)) --- N2_0((2))
    N1 --- N9_0((9))
    N2_0 --- N3_0((3))
    N2_0 --- N6_0((6))
    N9_0 --- N10_0((10))
    N9_0 --- N13_0((13))
    N3_0 --- N4_0((4))
    N3_0 --- N5_0((5))
    N6_0 --- N7_0((7))
    N6_0 --- N8_0((8))
    N10_0 --- N11_0((11))
    N10_0 --- N12_0((12))
    N13_0 --- N14_0((14))
    
```

Nodos visitados: 1, 2, 3, 6, 10, 13

Nodo actual: 13 (Vuelta de recursividad)

Datos de búsqueda:

- nivel = 2
- valorActual = 56
- pesoActual = 5
- solución:

0	1	2
1	1	0
- mejorSolucion:

1	0	1
0	1	2
- valorMejor = 72

2.La mochila 0-1

86

86

La mochila 0-1

Espacio de búsqueda

pesos

0	1	2
2	3	7

 valores

0	1	2
35	21	37

 MaxPeso: 10

Desanota

2.La mochila 0-1

87

La mochila 0-1

Espacio de búsqueda

pesos

0	1	2
2	3	7

 valores

0	1	2
35	21	37

 MaxPeso: 10

Vuelta de recursividad

2.La mochila 0-1

88

La mochila 0-1

Espacio de búsqueda

pesos

0	1	2
2	3	7

valores

0	1	2
35	21	37

MaxPeso: 10

Desarrollo

```

graph TD
    1((1)) -- 0 --> 3((3))
    1 -- 1 --> 6((6))
    3 -- 0 --> 4((4))
    3 -- 1 --> 5((5))
    6 -- 0 --> 7((7))
    6 -- 1 --> 8((8))
    9((9)) -- 0 --> 10((10))
    9 -- 1 --> 13((13))
    10 -- 0 --> 11((11))
    10 -- 1 --> 12((12))
    13 -- 0 --> 14((14))
    style 1 fill:#0070C0,stroke:#0070C0
    style 3 fill:#C00000
    style 6 fill:#C00000
    style 9 fill:#C00000
    style 10 fill:#C00000
    style 13 fill:#C00000
    style 14 fill:#C00000
    style 4 fill:#F0F0F0
    style 5 fill:#F0F0F0
    style 7 fill:#F0F0F0
    style 8 fill:#F0F0F0
    style 11 fill:#F0F0F0
    style 12 fill:#F0F0F0
  
```

nivel = 0
 valorActual = 0
 pesoActual = 0
 solución

0	0	0
0	1	2

 mejorSolucion

1	0	1
0	1	2

 valorMejor = 72

2.La mochila 0-1

89

La mochila 0-1

Espacio de búsqueda

pesos

0	1	2
2	3	7

valores

0	1	2
35	21	37

MaxPeso: 10

Solución FINAL

```

graph TD
    1((1)) -- 0 --> 2((2))
    1 -- 1 --> 9((9))
    2 -- 0 --> 3((3))
    2 -- 1 --> 6((6))
    9 -- 0 --> 10((10))
    9 -- 1 --> 13((13))
    10 -- 0 --> 11((11))
    10 -- 1 --> 12((12))
    style 1 fill:#C00000
    style 2 fill:#C00000
    style 9 fill:#C00000
    style 10 fill:#C00000
    style 13 fill:#C00000
    style 14 fill:#C00000
    style 3 fill:#F0F0F0
    style 6 fill:#F0F0F0
    style 11 fill:#F0F0F0
    style 12 fill:#F0F0F0
  
```

nivel = 0
 valorActual = 0
 pesoActual = 0
 solución

0	0	0
0	1	2

 mejorSolucion

1	0	1
0	1	2

 valorMejor = 72

2.La mochila 0-1

90

90

Cambio de monedas

Algunos problemas planteados

```

    graph TD
        A([Cambio de monedas]) --- B([Esquema selección óptima])
        B --- C([La mochila 0-1])
    
```

3.Cambio de monedas

91

91

Cambio de monedas

Enunciado

- Se dispone de un **sistema monetario** con suficientes monedas de cada tipo y de un importe a devolver C .
- El objetivo es obtener el desglose de monedas para pagar C , minimizando el número de monedas empleado.
 - Sea C la cantidad a devolver
 - Sea $V = (v_0, v_1, \dots, v_{n-1})$, los valores de los tipos de monedas del sistema monetario
 - solución: $\sum c_j v_j = C \ (\forall j=0..n-1)$, donde c_j es la cantidad de monedas de tipo j usadas en la solución.
 - Minimizar: $\sum c_j \ (\forall j=0..n-1)$

valores

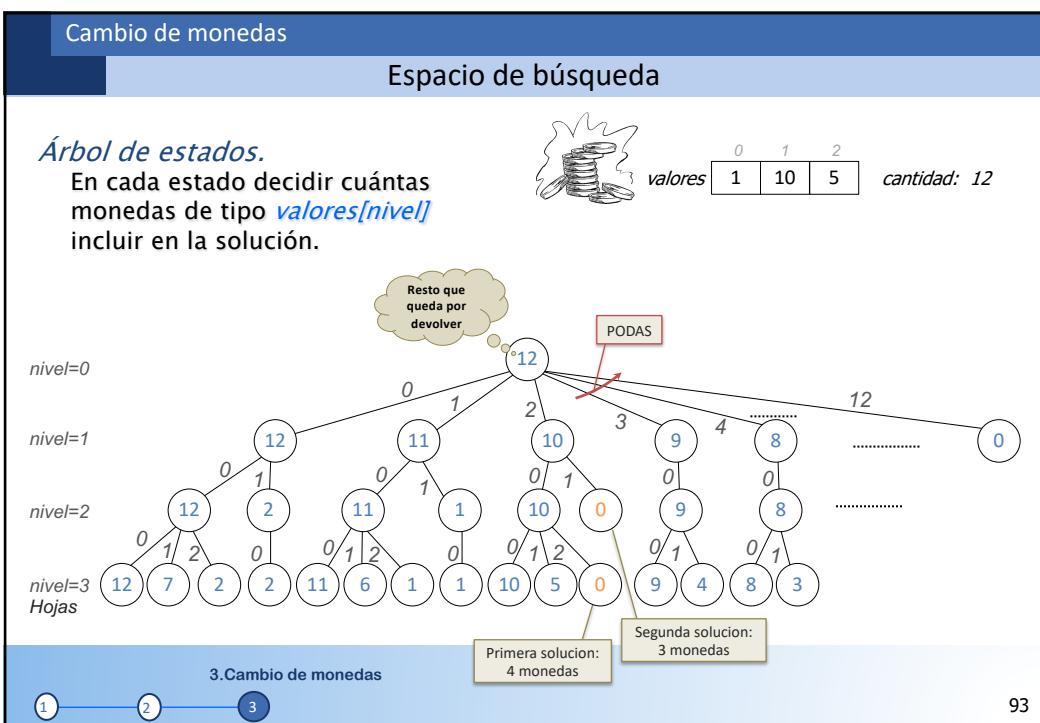
1	2	5	10	20	50
---	---	---	----	----	----

Cantidad a devolver: 42

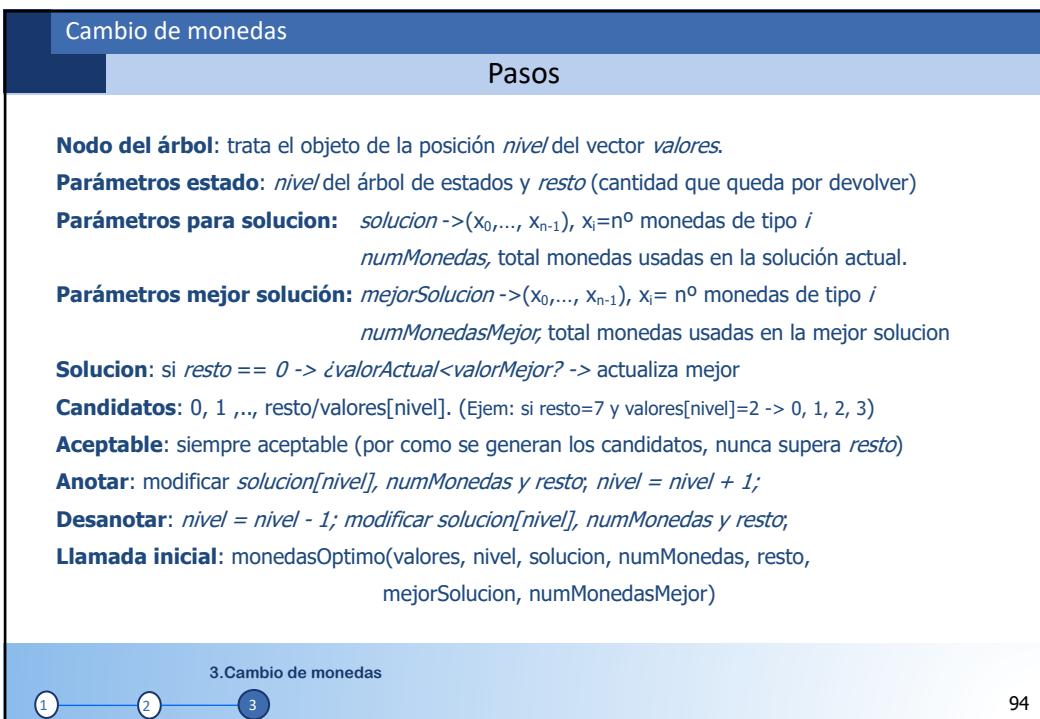
3.Cambio de monedas

92

92



93



94

Cambio de monedas

Implementación

```
void monedasOptimo(int[] valores, int nivel, int[] solucion, int numMonedas, int resto,
                     int[] mejorSolucion, Entero numMonedasMejor){
    if (resto==0){
        if (numMonedas<numMonedasMejor.getValor()){
            numMonedasMejor.setValor(numMonedas);
            for (int i=0; i<mejorSolucion.length;i++) mejorSolucion[i]=solucion[i];
        }
    } else if (nivel<valores.length){ //controla que no sea un nodo hoja
        for (int n=0; n <= (resto/valores[nivel]) ; n++){
            solucion[nivel]=n;
            numMonedas = numMonedas + n;
            resto = resto - valores[nivel]*n;
            nivel = nivel + 1;
            if (numMonedas<numMonedasMejor.getValor())
                monedasOptimo(valores, nivel, solucion, numMonedas,
                               resto, mejorSolucion, numMonedasMejor);
            nivel = nivel -1;
            resto = resto + valores[nivel]*n;
            numMonedas = numMonedas - n;
            solucion[nivel]=0;
        }
    }
}
```

3.Cambio de monedas



95

95

Cambio de monedas

Implementación

```
void monedasOptimo(int[] valores, int nivel, int[] solucion, int numMonedas, int resto,
                     int[] mejorSolucion, Entero numMonedasMejor){  
    if (resto==0){  
        if (numMonedas<numMonedasMejor.getValor()){  
            numMonedasMejor.setValor(numMonedas);  
            for (int i=0; i<mejorSolucion.length;i++) mejorSolucion[i]=solucion[i];  
        }  
    } else if (nivel<valores.length){ //controla que no sea un nodo hoja  
        for (int n=0; n <= (resto/valores[nivel]) ; n++){  
            solucion[nivel]=n;  
            numMonedas = numMonedas + n;  
            resto = resto - valores[nivel]*n;  
            nivel = nivel + 1;  
            if (numMonedas<numMonedasMejor.getValor())  
                monedasOptimo(valores, nivel, solucion, numMonedas,  
                               resto, mejorSolucion, numMonedasMejor);  
            nivel = nivel -1;  
            resto = resto + valores[nivel]*n;  
            numMonedas = numMonedas - n;  
            solucion[nivel]=0;  
        }  
    }  
}
```

Parámetros

3.Cambio de monedas



96

96

Cambio de monedas

Implementación

```

void monedasOptimo(int[] valores, int nivel, int[] solucion, int numMonedas, int resto,
                    int[] mejorSolucion, Entero numMonedasMejor){
    if (resto==0){
        if (numMonedas<numMonedasMejor.getValor()){
            numMonedasMejor.setValor(numMonedas);
            for (int i=0; i<mejorSolucion.length;i++) mejorSolucion[i]=solucion[i];
        }
    }
    else if (nivel<valores.length){ //controla que no sea un nodo hoja
        for (int n=0; n <= (resto/valores[nivel]) ; n++){
            solucion[nivel]=n;
            numMonedas = numMonedas + n;
            resto = resto - valores[nivel]*n;
            nivel = nivel + 1;
            if (numMonedas<numMonedasMejor.getValor())
                monedasOptimo(valores, nivel, solucion, numMonedas,
                               resto, mejorSolucion, numMonedasMejor);
            nivel = nivel -1;
            resto = resto + valores[nivel]*n;
            numMonedas = numMonedas - n;
            solucion[nivel]=0;
        }
    }
}

```

Solución

3.Cambio de monedas

97

97

Cambio de monedas

Implementación

```

void monedasOptimo(int[] valores, int nivel, int[] solucion, int numMonedas, int resto,
                    int[] mejorSolucion, Entero numMonedasMejor){
    if (resto==0){
        if (numMonedas<numMonedasMejor.getValor()){
            numMonedasMejor.setValor(numMonedas);
            for (int i=0; i<mejorSolucion.length;i++) mejorSolucion[i]=solucion[i];
        }
    }
    else if (nivel<valores.length){ //controla que no sea un nodo hoja
        for (int n=0; n <= (resto/valores[nivel]) ; n++){
            solucion[nivel]=n;
            numMonedas = numMonedas + n;
            resto = resto - valores[nivel]*n;
            nivel = nivel + 1;
            if (numMonedas<numMonedasMejor.getValor())
                monedasOptimo(valores, nivel, solucion, numMonedas,
                               resto, mejorSolucion, numMonedasMejor);
            nivel = nivel -1;
            resto = resto + valores[nivel]*n;
            numMonedas = numMonedas - n;
            solucion[nivel]=0;
        }
    }
}

```

Quedan candidatos:
Sólo para nodos
intermedios que no
sean solución (nunca
para nodos hoja)

3.Cambio de monedas

98

98

Cambio de monedas

Implementación

```

void monedasOptimo(int[] valores, int nivel, int[] solucion, int numMonedas, int resto,
                    int[] mejorSolucion, Entero numMonedasMejor){
    if (resto==0){
        if (numMonedas<numMonedasMejor.getValor()){
            numMonedasMejor.setValor(numMonedas);
            for (int i=0; i<mejorSolucion.length;i++) mejorSolucion[i]=solucion[i];
        }
    }
    else if (nivel<valores.length){ //controla que no sea un nodo hoja
        for (int n=0; n <= (resto/valores[nivel]) ; n++){
            solucion[nivel]=n;
            numMonedas = numMonedas + n;
            resto = resto - valores[nivel]*n;
            nivel = nivel + 1;
            if (numMonedas<numMonedasMejor.getValor())
                monedasOptimo(valores, nivel, solucion, numMonedas,
                               resto, mejorSolucion, numMonedasMejor);
            nivel = nivel -1;
            resto = resto + valores[nivel]*n;
            numMonedas = numMonedas - n;
            solucion[nivel]=0;
        }
    }
}

```

Candidatos

3.Cambio de monedas

99

99

Cambio de monedas

Implementación

```

void monedasOptimo(int[] valores, int nivel, int[] solucion, int numMonedas, int resto,
                    int[] mejorSolucion, Entero numMonedasMejor){
    if (resto==0){
        if (numMonedas<numMonedasMejor.getValor()){
            numMonedasMejor.setValor(numMonedas);
            for (int i=0; i<mejorSolucion.length;i++) mejorSolucion[i]=solucion[i];
        }
    }
    else if (nivel<valores.length){ //controla que no sea un nodo hoja
        for (int n=0; n <= (resto/valores[nivel]) ; n++){
            solucion[nivel]=n;
            numMonedas = numMonedas + n;
            resto = resto - valores[nivel]*n;
            nivel = nivel + 1;
            if (numMonedas<numMonedasMejor.getValor())
                monedasOptimo(valores, nivel, solucion, numMonedas,
                               resto, mejorSolucion, numMonedasMejor);
            nivel = nivel -1;
            resto = resto + valores[nivel]*n;
            numMonedas = numMonedas - n;
            solucion[nivel]=0;
        }
    }
}

```

Candidato aceptable
siempre por cómo
se generan los
candidatos

3.Cambio de monedas

100

100

Cambio de monedas

Implementación

```
void monedasOptimo(int[] valores, int nivel, int[] solucion, int numMonedas, int resto,
                     int[] mejorSolucion, Entero numMonedasMejor){
    if (resto==0){
        if (numMonedas<numMonedasMejor.getValor()){
            numMonedasMejor.setValor(numMonedas);
            for (int i=0; i<mejorSolucion.length;i++) mejorSolucion[i]=solucion[i];
        }
    } else if (nivel<valores.length){ //controla que no sea un nodo hoja
        for (int n=0; n <= (resto/valores[nivel]) ; n++){
            solucion[nivel]=n;
            numMonedas = numMonedas + n;
            resto = resto - valores[nivel]*n;
            nivel = nivel + 1;
            if (numMonedas<numMonedasMejor.getValor())
                monedasOptimo(valores, nivel, solucion, numMonedas,
                               resto, mejorSolucion, numMonedasMejor);
            nivel = nivel -1;
            resto = resto + valores[nivel]*n;
            numMonedas = numMonedas - n;
            solucion[nivel]=0;
        }
    }
}
```

anotar

3.Cambio de monedas

101

Cambio de monedas

Implementación

```
void monedasOptimo(int[] valores, int nivel, int[] solucion, int numMonedas, int resto,
                     int[] mejorSolucion, Entero numMonedasMejor){
    if (resto==0){
        if (numMonedas<numMonedasMejor.getValor()){
            numMonedasMejor.setValor(numMonedas);
            for (int i=0; i<mejorSolucion.length;i++) mejorSolucion[i]=solucion[i];
        }
    } else if (nivel<valores.length){ //controla que no sea un nodo hoja
        for (int n=0; n <= (resto/valores[nivel]) ; n++){
            solucion[nivel]=n;
            numMonedas = numMonedas + n;
            resto = resto - valores[nivel]*n;
            nivel = nivel + 1;
            if (numMonedas<numMonedasMejor.getValor())
                monedasOptimo(valores, nivel, solucion, numMonedas,
                               resto, mejorSolucion, numMonedasMejor);
            nivel = nivel -1;
            resto = resto + valores[nivel]*n;
            numMonedas = numMonedas - n;
            solucion[nivel]=0;
        }
    }
}
```

Avance en el espacio
de búsqueda. PODA

3.Cambio de monedas

102

Cambio de monedas

Implementación

```

void monedasOptimo(int[] valores, int nivel, int[] solucion, int numMonedas, int resto,
                     int[] mejorSolucion, Entero numMonedasMejor){
    if (resto==0){
        if (numMonedas<numMonedasMejor.getValor()){
            numMonedasMejor.setValor(numMonedas);
            for (int i=0; i<mejorSolucion.length;i++) mejorSolucion[i]=solucion[i];
        }
    } else if (nivel<valores.length){ //controla que no sea un nodo hoja
        for (int n=0; n <= (resto/valores[nivel]) ; n++){
            solucion[nivel]=n;
            numMonedas = numMonedas + n;
            resto = resto - valores[nivel]*n;
            nivel = nivel + 1;
            if (numMonedas<numMonedasMejor.getValor())
                monedasOptimo(valores, nivel, solucion, numMonedas,
                               resto, mejorSolucion, numMonedasMejor);
            nivel = nivel -1;
            resto = resto + valores[nivel]*n;
            numMonedas = numMonedas - n;
            solucion[nivel]=0;
        }
    }
}

```

desanotar

3.Cambio de monedas



103

103

Cambio de monedas

Implementación

```

void monedasOptimo(int[] valores, int nivel, int[] solucion, int numMonedas, int resto,
                     int[] mejorSolucion, Entero numMonedasMejor){
    if (resto==0){
        if (numMonedas<numMonedasMejor.getValor()){
            numMonedasMejor.setValor(numMonedas);
            for (int i=0; i<mejorSolucion.length;i++) mejorSolucion[i]=solucion[i];
        }
    } else if (nivel<valores.length){ //controla que no sea un nodo hoja
        for (int n=0; n <= (resto/valores[nivel]) ; n++){
            solucion[nivel]=n;
            numMonedas = numMonedas + n;
            resto = resto - valores[nivel]*n;
            nivel = nivel + 1;
            if (numMonedas<numMonedasMejor.getValor())
                monedasOptimo(valores, nivel, solucion, numMonedas,
                               resto, mejorSolucion, numMonedasMejor);
            nivel = nivel -1;
            resto = resto + valores[nivel]*n;
            numMonedas = numMonedas - n;
            solucion[nivel]=0;
        }
    }
}

int[] valores = {5, 10, 1}; Llamada inicial
int cantidad = 12;

int[] solucion = new int[valores.length];
for (int i=0; i<solucion.length; i++) solucion[i]=0;
int numMonedasActual = 0;
int resto = cantidad;

int[] mejorSolucion = new int[valores.length];
for (int i=0; i<mejorSolucion.length; i++) mejorSolucion[i]=0;
Entero numMonedasMejor = new Entero(Integer.MAX_VALUE);

monedasOptimo(valores, 0, solucion, numMonedasActual, resto,
               mejorSolucion, numMonedasMejor);

```



104

104