Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

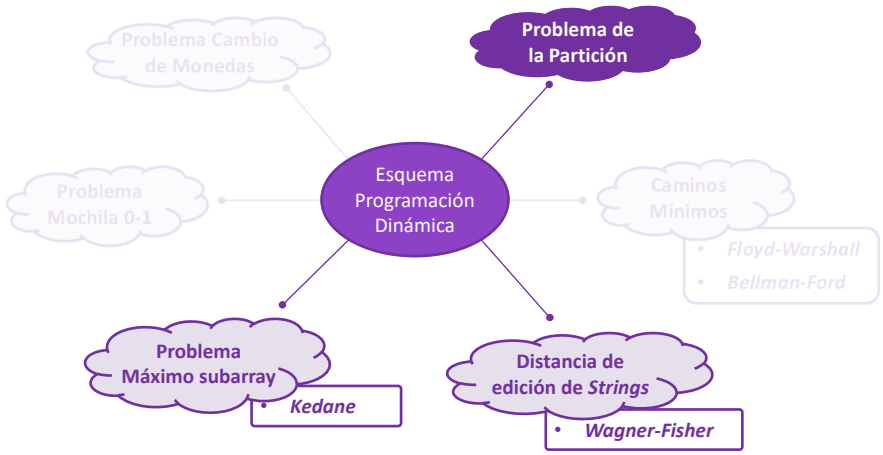
Tema 15. Ejemplos Algoritmos en Programación Dinámica

Algorítmica y Complejidad

1

Problema de la Partición

Algunos problemas planteados



2

Problema de la Partición

Algunos problemas planteados

Enunciado

Dado un conjunto de bienes valorados

2	4	1	7	10	6	3	5

Repartir los bienes entre dos personas de manera que obtengan ambos bienes por el mismo valor

1. Problema de la Partición

1 2 3

3

Problema de la Partición

Enunciado

Enunciado

Dado un conjunto de bienes valorados

2	4	1	7	10	6	3	5

Repartir los bienes entre dos personas de manera que obtengan ambos bienes por el mismo valor

2	1	7	3	5

€

18

4	10	6

€

20

1. Problema de la Partición

1 2 3

4

Problema de la Partición

Enunciado

Enunciado

Dado un conjunto de bienes valorados

2	4	1	7	10	6	3	5

Repartir los bienes entre dos personas de manera que obtengan ambos bienes por el mismo valor

4	1	10	€

15

2	7	6	3	5	€

23

1. Problema de la Partición

1 — 2 — 3

5

Problema de la Partición

Enunciado

Enunciado

Dado un conjunto de bienes valorados

2	4	1	7	10	6	3	5

Repartir los bienes entre dos personas de manera que obtengan ambos bienes por el mismo valor

4	7	3	5	€

19

2	1	10	6	€

19

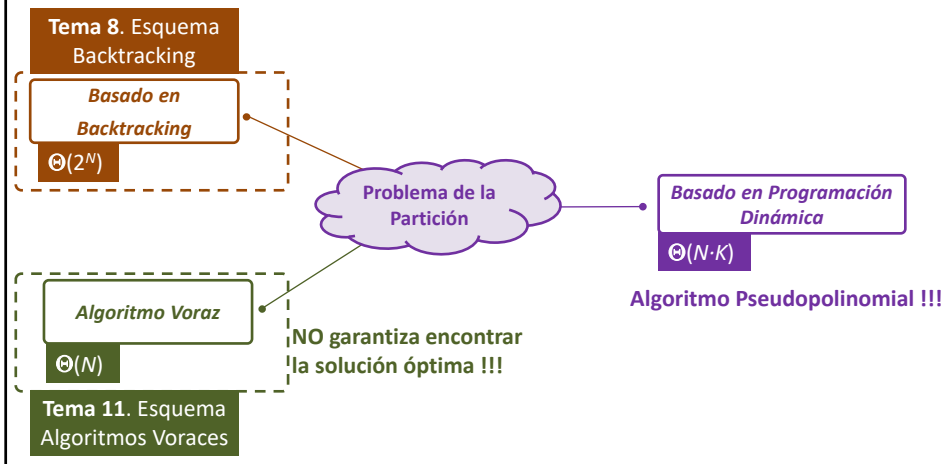
1. Problema de la Partición

1 — 2 — 3

6

Problema de la Partición

Algoritmos de Resolución



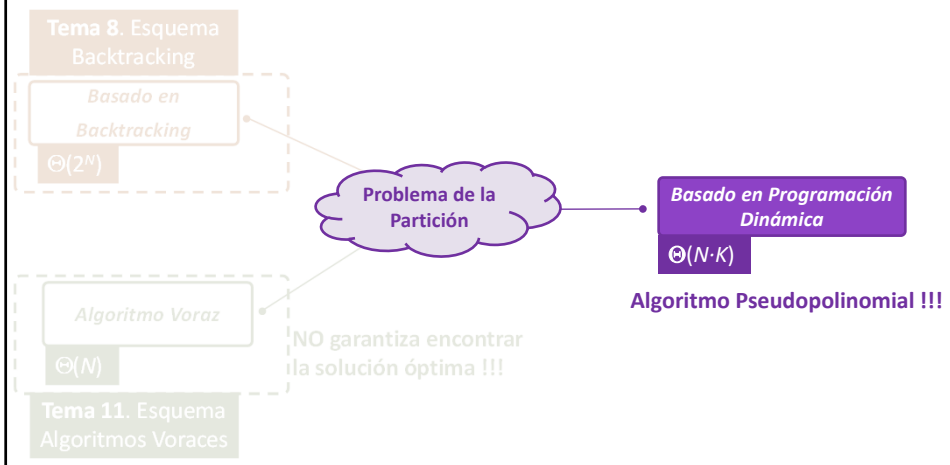
1. Problema de la Partición

1 — 2 — 3

7

Problema de la Partición

Algoritmos de Resolución



1. Problema de la Partición

1 — 2 — 3

8

Problema de la Partición

Relación de recurrencia

j:3

i:7 bienes: | 4 | 5 | 6 | 7 | 8 | €

hayReparto(i,j): Es posible conseguir que la persona #0 consiga bienes con valor total *i* utilizando los bienes 0,1,…,j

9

Problema de la Partición

Relación de recurrencia

j:4

i:15 bienes: | 4 | 5 | 6 | 7 | 8 | €

hayReparto(15,4): Es posible conseguir que la persona **#0** consiga bienes con valor total **15** utilizando los bienes 0,1,2,3,4

Problema de la partición !!

10

Problema de la Partición

Relación de recurrencia

j:3

i:7 bienes:

4	5	6	7	8

€

hayReparto(7,3): Es posible conseguir que la persona **#0** consiga bienes con valor total **7** utilizando los bienes **0,1,2,3**

11

Problema de la Partición

Relación de recurrencia

j:2

i:7 bienes:

4	5	6	7	8

€

hayReparto(7,2): Es posible conseguir que la persona **#0** consiga bienes con valor total **7** utilizando los bienes **0,1,2**

12

Problema de la Partición

Relación de recurrencia

j:1

i:4 bienes: | 4 | 5 | 6 | 7 | 8 | €

hayReparto(4,1): Es posible conseguir que la persona **#0** consiga bienes con valor total **4** utilizando los bienes **0,1**

13

Problema de la Partición

Relación de recurrencia

j:0

i:4 bienes: | 4 | 5 | 6 | 7 | 8 | €

hayReparto(4,0): Es posible conseguir que la persona **#0** consiga bienes con valor total **4** utilizando el bien **0**

Caso Base:

$$\boldsymbol{hayReparto}(i,0) = \begin{cases} \textbf{True} & si\ bienes[0] = i \\ \textbf{True} & si\ i = 0 \\ \textbf{False} & En\ otro\ caso \end{cases}$$

14

Problema de la Partición

Relación de recurrencia

j:4

i:15 bienes: | 4 | 5 | 6 | 7 | 8 | €

hayReparto(i,j): Es posible conseguir que la persona **#0** consiga bienes con valor total i utilizando los bienes 0,1,…,j

Caso Recursivo:

$$\boldsymbol{hayReparto}(i,j) = \begin{cases} \boldsymbol{True} & si\ \boldsymbol{hayReparto}(i,j-1) = True \\ \boldsymbol{True} & si\ \boldsymbol{hayReparto}(i - bienes[j], j-1) = True \\ \boldsymbol{False} & En\ otro\ caso \end{cases}$$

15

Problema de la Partición

Relación de recurrencia

j:4

i:15 bienes: | 4 | 5 | 6 | 7 | 8 | €

hayReparto(i,j): Es posible conseguir que la persona **#0** consiga bienes con valor total i utilizando los bienes 0,1,…,j

j-1

i:15 | 4 | 5 | 6 | 7 | 8 |

Caso Recursivo:

$$\boldsymbol{hayReparto}(i,j) = \begin{cases} \boldsymbol{True} & si\ \boldsymbol{hayReparto}(i,j-1) = True \\ \boldsymbol{True} & si\ \boldsymbol{hayReparto}(i - bienes[j], j-1) = True \\ \boldsymbol{False} & En\ otro\ caso \end{cases}$$

16

Problema de la Partición

Relación de recurrencia

j:4

i:15 bienes: | 4 | 5 | 6 | 7 | 8 | €

hayReparto(i,j): Es posible conseguir que la persona **#0** consiga bienes con valor total i utilizando los bienes 0,1,…,j

Caso Recursivo:

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

j-1

i:7 | 4 | 5 | 6 | 7 | 8 |

17

Problema de la Partición

Relación de recurrencia

j:4

i:15 bienes: | 4 | 5 | 6 | 7 | 8 | €

hayReparto(i,j): Es posible conseguir que la persona **#0** consiga bienes con valor total i utilizando los bienes 0,1,…,j

Caso Recursivo:

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

1. Problema de la Partición
1 2 3

18

Problema de la Partición

Relación de recurrencia

j:0

i:1 bienes: 4 5 6 7 8 €

$$hayReparto(i,0) = \begin{cases} True & si\ bienes[0] = i \\ True & si\ i = 0 \\ False & En\ otro\ caso \end{cases}$$

hayReparto(0,0)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T															
1																
2																
3																
4																

1. Problema de la Partición

1 2 3

19

Problema de la Partición

Relación de recurrencia

j:0

i:1 bienes: 4 5 6 7 8 €

$$hayReparto(i,0) = \begin{cases} True & si\ bienes[0] = i \\ True & si\ i = 0 \\ False & En\ otro\ caso \end{cases}$$

hayReparto(1,0)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F														
1																
2																
3																
4																

1. Problema de la Partición

1 2 3

20

Problema de la Partición

Relación de recurrencia

j:0

i:2 bienes: 4 5 6 7 8 €

$$hayReparto(i,0) = \begin{cases} True & si\ bienes[0] = i \\ True & si\ i = 0 \\ False & En\ otro\ caso \end{cases}$$

hayReparto(2,0)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F													
1																
2																
3																
4																

21

Problema de la Partición

Relación de recurrencia

j:0

i:3 bienes: 4 5 6 7 8 €

$$hayReparto(i,0) = \begin{cases} True & si\ bienes[0] = i \\ True & si\ i = 0 \\ False & En\ otro\ caso \end{cases}$$

hayReparto(3,0)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F												
1																
2																
3																
4																

22

Problema de la Partición

Relación de recurrencia

j:0

i:4 bienes: 4 5 6 7 8 €

$$hayReparto(i,0) = \begin{cases} True & si\ bienes[0] = i \\ True & si\ i = 0 \\ False & En\ otro\ caso \end{cases}$$

hayReparto(4,0)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T											
1																
2																
3																
4																

1. Problema de la Partición

1 2 3

23

Problema de la Partición

Relación de recurrencia

j:0

i:5 bienes: 4 5 6 7 8 €

$$hayReparto(i,0) = \begin{cases} True & si\ bienes[0] = i \\ True & si\ i = 0 \\ False & En\ otro\ caso \end{cases}$$

hayReparto(5,0)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F										
1																
2																
3																
4																

1. Problema de la Partición

1 2 3

24

Problema de la Partición

Relación de recurrencia

j:0

i:10 bienes: 4 5 6 7 8 €

$$hayReparto(i,0) = \begin{cases} True & si\ bienes[0] = i \\ True & si\ i = 0 \\ False & En\ otro\ caso \end{cases}$$

hayReparto(10,0)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F					
1																
2																
3																
4																

1. Problema de la Partición

1 2 3

25

Problema de la Partición

Relación de recurrencia

j:0

i:15 bienes: 4 5 6 7 8 €

$$hayReparto(i,0) = \begin{cases} True & si\ bienes[0] = i \\ True & si\ i = 0 \\ False & En\ otro\ caso \end{cases}$$

hayReparto(15,0)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1																
2																
3																
4																

1. Problema de la Partición

1 2 3

26

Problema de la Partición

Relación de recurrencia

j:1

i:0 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(0,1)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T															
2																
3																
4																

27

Problema de la Partición

Relación de recurrencia

j:1

i:1 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(1,1)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F														
2																
3																
4																

28

Problema de la Partición

Relación de recurrencia

j:1

i:2 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(2,1)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F													
2																
3																
4																

29

Problema de la Partición

Relación de recurrencia

j:1

i:3 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(3,1)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F												
2																
3																
4																

30

Problema de la Partición

Relación de recurrencia

j:1

i:4 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(4,1)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T											
2																
3																
4																

31

Problema de la Partición

Relación de recurrencia

j:1

i:5 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(5,1)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T										
2																
3																
4																

32

Problema de la Partición

Relación de recurrencia

j:1

i:6 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(6,1)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F									
2																
3																
4																

33

Problema de la Partición

Relación de recurrencia

j:1

i:7 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(7,1)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F								
2																
3																
4																

34

Problema de la Partición

Relación de recurrencia

j:1

i:9 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(8,1)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F							
2																
3																
4																

35

Problema de la Partición

Relación de recurrencia

j:1

i:9 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(9,1)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T						
2																
3																
4																

36

Problema de la Partición

Relación de recurrencia

j:1

i:10 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

i

hayReparto(10,1)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F					
2																
3																
4																

1. Problema de la Partición

1 2 3

37

Problema de la Partición

Relación de recurrencia

j:1

i:15 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

i

hayReparto(15,1)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2																
3																
4																

1. Problema de la Partición

1 2 3

38

Problema de la Partición

Relación de recurrencia

j:2

i:0 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(0,2)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2	T															
3																
4																

39

Problema de la Partición

Relación de recurrencia

j:2

i:1 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(1,2)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2	T	F														
3																
4																

40

Problema de la Partición

Relación de recurrencia

j:2

i:2 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(2,2)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2	T	F	F													
3																
4																

1. Problema de la Partición

1 2 3

41

Problema de la Partición

Relación de recurrencia

j:2

i:3 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(3,2)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2	T	F	F	F												
3																
4																

1. Problema de la Partición

1 2 3

42

Problema de la Partición

Relación de recurrencia

j:2

i:4 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(4,2)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2	T	F	F	F	T											
3																
4																

1. Problema de la Partición

1 2 3

43

Problema de la Partición

Relación de recurrencia

j:2

i:5 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(5,2)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2	T	F	F	F	T	T										
3																
4																

1. Problema de la Partición

1 2 3

44

Problema de la Partición

Relación de recurrencia

j:2

i:6 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(6,2)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2	T	F	F	F	T	T	T									
3																
4																

45

Problema de la Partición

Relación de recurrencia

j:2

i:7 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(7,2)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2	T	F	F	F	T	T	T	F								
3																
4																

46

Problema de la Partición

Relación de recurrencia

j:2

i:8 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(8,2)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2	T	F	F	F	T	T	T	F	F							
3																
4																

1. Problema de la Partición

1 2 3

47

Problema de la Partición

Relación de recurrencia

j:2

i:9 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(9,2)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2	T	F	F	F	T	T	T	F	F	T						
3																
4																

1. Problema de la Partición

1 2 3

48

Problema de la Partición

Relación de recurrencia

j:2

i:10 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i,j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(10,2)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2	T	F	F	F	T	T	T	F	F	T	T					
3																
4																

49

Problema de la Partición

Relación de recurrencia

j:2

i:11 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i,j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(11,2)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2	T	F	F	F	T	T	T	F	F	T	T	T				
3																
4																

50

Problema de la Partición

Relación de recurrencia

j:2

i:15 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i,j-1) = True \\ True & si\ hayReparto(i-bienes[j],j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(15,2)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2	T	F	F	F	T	T	T	F	F	T	T	T	F	F	F	T
3																
4																

51

Problema de la Partición

Relación de recurrencia

j:3

i:8 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i,j-1) = True \\ True & si\ hayReparto(i-bienes[j],j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(8,3)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2	T	F	F	F	T	T	T	F	F	T	T	T	F	F	F	T
3	T	F	F	F	T	T	T	T	F							
4																

52

Problema de la Partición

Relación de recurrencia

j:3

i:13 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(13,3)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2	T	F	F	F	T	T	T	F	F	T	T	T	F	F	F	T
3	T	F	F	F	T	T	T	T	F	T	T	T	T	T		
4																

53

Problema de la Partición

Relación de recurrencia

j:3

i:15 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(15,3)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2	T	F	F	F	T	T	T	F	F	T	T	T	F	F	F	T
3	T	F	F	F	T	T	T	T	F	T	T	T	T	T	F	T
4																

54

Problema de la Partición

Relación de recurrencia

j:4

i:3 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i,j-1) = True \\ True & si\ hayReparto(i-bienes[j],j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(3,4)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2	T	F	F	F	T	T	T	F	F	T	T	T	F	F	F	T
3	T	F	F	F	T	T	T	T	F	T	T	T	T	T	F	T
4	T	F	F	F												

1. Problema de la Partición

1 2 3

55

Problema de la Partición

Relación de recurrencia

j:4

i:8 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i,j-1) = True \\ True & si\ hayReparto(i-bienes[j],j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(8,4)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2	T	F	F	F	T	T	T	F	F	T	T	T	F	F	F	T
3	T	F	F	F	T	T	T	T	F	T	T	T	T	T	F	T
4	T	F	F	F	T	T	T	T	T							

1. Problema de la Partición

1 2 3

56

Problema de la Partición

Relación de recurrencia

j:4

i:15 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

i

hayReparto(15,4)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
j 0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2	T	F	F	F	T	T	T	F	F	T	T	T	F	F	F	T
3	T	F	F	F	T	T	T	T	F	T	T	T	T	T	F	T
4	T	F	F	F	T	T	T	T	T	T	T	T	T	T	T	T

1. Problema de la Partición

1 2 3

57

Problema de la Partición

Relación de recurrencia

Actividad 15.1. Implementa el algoritmo que determina si hay reparto equitativo usando programación dinámica.

```
boolean hayRepartoEquitativo (int[] bienes){
  int objetivo=sumaBienes(bienes)/2;
  boolean[][] hayReparto = new boolean[objetivo+1][bienes.length];
  hayReparto[0][0]= true;
  for (int i=1;i<=objetivo; i++)
     hayReparto[i][0]=(bienes[0]==i);
  for (int j=1;j<bienes.length; j++)
    for (int i=0;i<=objetivo; i++)
      if (hayReparto[i][j-1])
         hayReparto[i][j]=true;
      else if (bienes[j]<=i && hayReparto[i-bienes[j]][j-1])
         hayReparto[i][j]= true;
      else
         hayReparto[i][j]=false;
  return hayReparto[objetivo][bienes.length-1];
}
```

1 2 3

58

Problema de la Partición

Relación de recurrencia

Actividad 15.1. **Implementa el algoritmo que determina si hay reparto equitativo usando programación dinámica.**

```
boolean hayRepartoEquitativo (int[] bienes){
  int objetivo=sumaBienes(bienes)/2;
  boolean[][] hayReparto = new boolean[objetivo+1][bienes.length];
  hayReparto[0][0]= true;
  for (int i=1;i<=objetivo; i++) {
     hayReparto[i][0]=(bienes[0]==i);
  for (int j=1;j<bienes.length; j++)
  for (int i=0;i<=objetivo; i++)
        hayReparto[i][j]=false;
  return hayReparto[objetivo][bienes.length-1];
}
```

Caso Base:

$$\boldsymbol{hayReparto}(\boldsymbol{i},0) = \begin{cases} \boldsymbol{True} & si\ bienes[0] = \boldsymbol{i} \\ \boldsymbol{True} & si\ \boldsymbol{i} = 0 \\ \boldsymbol{False} & En\ otro\ caso \end{cases}$$

1 2 3

59

Problema de la Partición

Relación de recurrencia

Actividad 15.1. **Implementa el algoritmo que determina si hay reparto equitativo usando programación dinámica.**

Caso Recursivo:

```
quitativo (int[] bienes){
```

$$\boldsymbol{hayReparto}(\boldsymbol{i},\boldsymbol{j}) = \begin{cases} \boldsymbol{True} & si\ \boldsymbol{hayReparto}(\boldsymbol{i},\boldsymbol{j}-1) = True \\ \boldsymbol{True} & si\ \boldsymbol{hayReparto}(\boldsymbol{i}-bienes[\boldsymbol{j}],\boldsymbol{j}-1) = True \\ \boldsymbol{False} & En\ otro\ caso \end{cases}$$

```
  for (int i=0;i<=objetivo; i++)
      if (hayReparto[i][j-1])
         hayReparto[i][j]=true;
      else if (bienes[j]<=i && hayReparto[i-bienes[j]][j-1])
         hayReparto[i][j]= true;
      else
         hayReparto[i][j]=false;
  return hayReparto[objetivo][bienes.length-1];
}
```

1 2 3

60

Problema de la Partición

Relación de recurrencia

Actividad 15.2. **Calcula la complejidad del algoritmo en tiempo según el número de bienes, *N*, y el valor del reparto equitativo, *K* (*variable* `objetivo`).**

```
boolean hayRepartoEquitativo (int[] bienes){
  int objetivo=sumaBienes(bienes)/2;
  boolean[][] hayReparto = new boolean[objetivo+1][bienes.length];
  hayReparto[0][0]= true;
  for (int i=1;i<=objetivo; i++)
     hayReparto[i][0]=(bienes[0]==i);
  for (int j=1;j<bienes.length; j++)
    for (int i=0;i<=objetivo; i++)
      if (hayReparto[i][j-1])
         hayReparto[i][j]=true;
      else if (bienes[j]<=i && hayReparto[i-bienes[j]][j-1])
         hayReparto[i][j]= true;
      else
         hayReparto[i][j]=false;
  return hayReparto[objetivo][bienes.length-1];
}
```

$\Theta(N \cdot K)$

1 — 2 — 3

61

Problema de la Partición

Relación de recurrencia

Actividad 15.3. **Calcula la complejidad del algoritmo en tiempo según el número de bits, k, necesarios para almacenar el valor del reparto equitativo, *K* (*variable* `objetivo`).**

```
boolean hayRepartoEquitativo (int[] bienes){
  int objetivo=sumaBienes(bienes)/2;
  boolean[][] hayReparto = new boolean[objetivo+1][bienes.length];
  hayReparto[0][0]= true;
  for (int i=1;i<=objetivo; i++)
     hayReparto[i][0]=(bienes[0]==i);
  for (int j=1;j<bienes.length; j++)
    for (int i=0;i<=objetivo; i++)
      if (hayReparto[i][j-1])
         hayReparto[i][j]=true;
      else if (bienes[j]<=i && hayReparto[i-bienes[j]][j-1])
         hayReparto[i][j]= true;
      else
         hayReparto[i][j]=false;
  return hayReparto[objetivo][bienes.length-1];
}
```

$\Theta(N \cdot K)$

$\Theta(N \cdot 2^k)$

Algoritmo pseudopolinomial

1 — 2 — 3

62

Problema de la Partición

Relación de recurrencia

Actividad 15.4. Calcula la complejidad del algoritmo en memoria según el número de bienes, *N*, y el valor del reparto equitativo, *K* (*variable* `objetivo`).

$\Theta(N \cdot K)$

K=15

`hayReparto(15,4)`

N=4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2	T	F	F	F	T	T	T	F	F	T	T	T	F	F	F	T
3	T	F	F	F	T	T	T	T	F	T	T	T	T	T	F	T
4	T	F	F	F	T	T	T	T	T	T	T	T	T	T	T	T

63

Problema de la Partición

Reduciendo la Complejidad en Memoria

`j:3`

`i:8` `bienes:` 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

i

`hayReparto(8,3)`

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2	T	F	F	F	T	T	T	F	F	T	T	T	F	F	F	T
3	T	F	F	F	T	T	T	T	F							
4																

64

Problema de la Partición

Reduciendo la Complejidad en Memoria

j:3

i:8 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(8,3)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F
2	T	F	F	F	T	T	T	F	F	T	T	T	F	F	F	T
3	T	F	F	F	T	T	T	T	F							
4																

No necesitamos ya estos valores

Objetivo: Calcular este valor

1. Problema de la Partición

1 2 3

65

Problema de la Partición

Reduciendo la Complejidad en Memoria

j:0

i:10 bienes: 4 5 6 7 8 €

$$hayReparto(i,0) = \begin{cases} True & si\ bienes[0] = i \\ True & si\ i = 0 \\ False & En\ otro\ caso \end{cases}$$

hayReparto(10,0)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
j=0 0	T	F	F	F	T	F	F	F	F	F	F					
1																

1. Problema de la Partición

1 2 3

66

Problema de la Partición

Reduciendo la Complejidad en Memoria

j:1

i:6 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(6,1)

i

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
j=0	0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
j=1	1	T	F	F	F	T	T	F									

1. Problema de la Partición

1 2 3

67

Problema de la Partición

Reduciendo la Complejidad en Memoria

j:2

i:9 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(9,2)

i

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
j=2	0	T	F	F	F	T	T	T	F	F	T						
j=1	1	T	F	F	F	T	T	F	F	F	T	F	F	F	F	F	F

1. Problema de la Partición

1 2 3

68

Problema de la Partición

Reduciendo la Complejidad en Memoria

j:3

i:8 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i,j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(8,3)

i

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
j=2	0	T	F	F	F	T	T	T	F	F	T	T	T	F	F	F	T
j=3	1	T	F	F	F	T	T	T	T	F							

1. Problema de la Partición

1 2 3

69

Problema de la Partición

Reduciendo la Complejidad en Memoria

j:4

i:15 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i,j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(15,4)

i

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
j=4	0	T	F	F	F	T	T	T	T	T	T	T	T	T	T	T	T
j=3	1	T	F	F	F	T	T	T	T	F	T	T	T	T	T	F	T

1. Problema de la Partición

1 2 3

70

Problema de la Partición

Reduciendo la Complejidad en Memoria

Actividad 15.5. Optimiza en memoria el algoritmo anterior que determina si hay reparto equitativo usando programación dinámica.

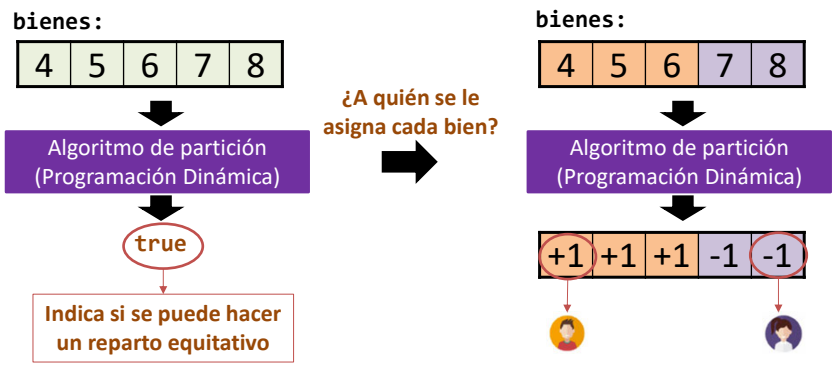
```
boolean hayRepartoEquitativo (int[] bienes){
  int objetivo=sumaBienes(bienes)/2;
  boolean[][] hayReparto = new boolean[objetivo+1][2];
  hayReparto[0][0]= true;
  for (int i=1;i<=objetivo; i++)
     hayReparto[i][0]=(bienes[0]==i);
  for (int j=1;j<bienes.length; j++)
    for (int i=0;i<=objetivo; i++)
      if (hayReparto[i][(j-1) % 2])
         hayReparto[i][j % 2]=true;
      else if (bienes[j]<=i && hayReparto[i-bienes[j]][(j-1) % 2])
         hayReparto[i][j % 2]= true;
      else
         hayReparto[i][j % 2]=false;
  return hayReparto[objetivo][(bienes.length-1) % 2];
}
```

1 — 2 — 3

71

Problema de la Partición

Conociendo la decisión



1. Problema de la Partición

1 — 2 — 3

72

Problema de la Partición

Relación de recurrencia

bienes: 4 5 6 7 8

Algoritmo de partición (Programación Dinámica)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-1	0	0	0	+1	0	0	0	0	0	0	0	0	0	0	0
1	-1	0	0	0	-1	+1	0	0	0	+1	0	0	0	0	0	0
2	-1	0	0	0	-1	-1	+1	0	0	-1	+1	+1	0	0	0	+1
3	-1	0	0	0	-1	-1	-1	+1	0	-1	-1	-1	-1	-1	0	-1
4	-1	0	0	0	-1	-1	-1	-1	+1	-1	-1	-1	-1	-1	+1	-1

1. Problema de la Partición

1 — 2 — 3

73

Problema de la Partición

Relación de recurrencia

j:2

i:15 bienes: 4 5 6 7 8

Algoritmo de partición (Programación Dinámica)

El bien **2** se le asigna a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-1	0	0	0	+1	0	0	0	0	0	0	0	0	0	0	0
1	-1	0	0	0	-1	+1	0	0	0	+1	0	0	0	0	0	0
2	-1	0	0	0	-1	-1	+1	0	0	-1	+1	+1	0	0	0	+1
3	-1	0	0	0	-1	-1	-1	+1	0	-1	-1	-1	-1	-1	0	-1
4	-1	0	0	0	-1	-1	-1	-1	+1	-1	-1	-1	-1	-1	+1	-1

1. Problema de la Partición

1 — 2 — 3

74

Problema de la Partición

Relación de recurrencia

j:2

i:4 bienes: 4 5 6 7 8

El bien **2** se le asigna a

Algoritmo de partición (Programación Dinámica)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-1	0	0	0	+1	0	0	0	0	0	0	0	0	0	0	0
1	-1	0	0	0	-1	+1	0	0	0	+1	0	0	0	0	0	0
2	-1	0	0	0	-1	-1	+1	0	0	-1	+1	+1	0	0	0	+1
3	-1	0	0	0	-1	-1	-1	+1	0	-1	-1	-1	-1	-1	0	-1
4	-1	0	0	0	-1	-1	-1	-1	+1	-1	-1	-1	-1	-1	+1	-1

1. Problema de la Partición

1 2 3

75

Problema de la Partición

Relación de recurrencia

j:2

i:3 bienes: 4 5 6 7 8

No es posible el reparto

Algoritmo de partición (Programación Dinámica)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-1	0	0	0	+1	0	0	0	0	0	0	0	0	0	0	0
1	-1	0	0	0	-1	+1	0	0	0	+1	0	0	0	0	0	0
2	-1	0	0	0	-1	-1	+1	0	0	-1	+1	+1	0	0	0	+1
3	-1	0	0	0	-1	-1	-1	+1	0	-1	-1	-1	-1	-1	0	-1
4	-1	0	0	0	-1	-1	-1	-1	+1	-1	-1	-1	-1	-1	+1	-1

1. Problema de la Partición

1 2 3

76

Problema de la Partición

Conociendo la decisión: Caso Base

j:0

i:10 bienes: 4 5 6 7 8 €

$$hayReparto(i,0) = \begin{cases} True & si\ bienes[0] = i \\ True & si\ i = 0 \\ False & En\ otro\ caso \end{cases}$$

i

hayReparto(10,0)

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F					
1																
2																
3																
4																

1. Problema de la Partición

1 2 3

77

Problema de la Partición

Conociendo la decisión: Caso Base

j:0

i:10 bienes: 4 5 6 7 8 €

$$hayReparto(i,0) = \begin{cases} +1 & si\ bienes[0] = i \\ -1 & si\ i = 0 \\ 0 & En\ otro\ caso \end{cases}$$

hayReparto(10,0)

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-1	0	0	0	+1	0	0	0	0	0	0					
1																
2																
3																
4																

1. Problema de la Partición

1 2 3

78

Problema de la Partición

Conociendo la decisión: Caso Recursivo

j:1

i:9 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} True & si\ hayReparto(i, j-1) = True \\ True & si\ hayReparto(i - bienes[j], j-1) = True \\ False & En\ otro\ caso \end{cases}$$

hayReparto(9,1)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	T	F	F	F	T	F	F	F	F	F	F	F	F	F	F	F
1	T	F	F	F	T	T	F	F	F	T						
2																
3																
4																

1. Problema de la Partición

1 2 3

79

Problema de la Partición

Conociendo la decisión: Caso Recursivo

j:1

i:9 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} -1 & si\ hayReparto(i, j-1) \neq 0 \\ +1 & si\ hayReparto(i - bienes[j], j-1) \neq 0 \\ 0 & En\ otro\ caso \end{cases}$$

hayReparto(9,1)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-1	0	0	0	+1	0	0	0	0	0	0	0	0	0	0	0
1	-1	0	0	0	-1	+1	0	0	0	+1						
2																
3																
4																

1. Problema de la Partición

1 2 3

80

Problema de la Partición

Conociendo la decisión: Caso Recursivo

j:4

i:15 bienes: 4 5 6 7 8 €

$$hayReparto(i,j) = \begin{cases} -1 & si\ hayReparto(i,j-1) \neq 0 \\ +1 & si\ hayReparto(i-bienes[j],j-1) \neq 0 \\ 0 & En\ otro\ caso \end{cases}$$

hayReparto(15,4)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-1	0	0	0	+1	0	0	0	0	0	0	0	0	0	0	0
1	-1	0	0	0	-1	+1	0	0	0	+1	0	0	0	0	0	0
2	-1	0	0	0	-1	-1	+1	0	0	-1	+1	+1	0	0	0	+1
3	-1	0	0	0	-1	-1	-1	+1	0	-1	-1	-1	+1	+1	0	-1
4	-1	0	0	0	-1	-1	-1	-1	+1	-1	-1	-1	-1	-1	+1	-1

1. Problema de la Partición

1 2 3

81

Problema de la Partición

Conociendo la decisión

j:4

i:15 bienes: 4 5 6 7 8 €

hayReparto(15,4)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-1	0	0	0	+1	0	0	0	0	0	0	0	0	0	0	0
1	-1	0	0	0	-1	+1	0	0	0	+1	0	0	0	0	0	0
2	-1	0	0	0	-1	-1	+1	0	0	-1	+1	+1	0	0	0	+1
3	-1	0	0	0	-1	-1	-1	+1	0	-1	-1	-1	+1	+1	0	-1
4	-1	0	0	0	-1	-1	-1	-1	+1	-1	-1	-1	-1	-1	+1	-1

1. Problema de la Partición

1 2 3

82

Problema de la Partición

Conociendo la decisión

j:3

i:15 bienes: 4 5 6 7 8 €

i

hayReparto(15,3)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-1	0	0	0	+1	0	0	0	0	0	0	0	0	0	0	0
1	-1	0	0	0	-1	+1	0	0	0	+1	0	0	0	0	0	0
2	-1	0	0	0	-1	-1	+1	0	0	-1	+1	+1	0	0	0	+1
3	-1	0	0	0	-1	-1	-1	+1	0	-1	-1	-1	+1	+1	0	-1
4	-1	0	0	0	-1	-1	-1	-1	+1	-1	-1	-1	-1	-1	+1	-1

1. Problema de la Partición

1 2 3

83

Problema de la Partición

Conociendo la decisión

j:2

i:15 bienes: 4 5 6 7 8 €

15-6=9

i

hayReparto(15,3)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-1	0	0	0	+1	0	0	0	0	0	0	0	0	0	0	0
1	-1	0	0	0	-1	+1	0	0	0	+1	0	0	0	0	0	0
2	-1	0	0	0	-1	-1	+1	0	0	-1	+1	+1	0	0	0	+1
3	-1	0	0	0	-1	-1	-1	+1	0	-1	-1	-1	+1	+1	0	-1
4	-1	0	0	0	-1	-1	-1	-1	+1	-1	-1	-1	-1	-1	+1	-1

1. Problema de la Partición

1 2 3

84

Problema de la Partición

Conociendo la decisión

j:1

i:9 bienes: 4 5 6 7 8 €

9-5=4

i

hayReparto(9,1)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-1	0	0	0	+1	0	0	0	0	0	0	0	0	0	0	0
1	-1	0	0	0	-1	+1	0	0	0	+1	0	0	0	0	0	0
2	-1	0	0	0	-1	-1	+1	0	0	-1	+1	+1	0	0	0	+1
3	-1	0	0	0	-1	-1	-1	+1	0	-1	-1	-1	+1	+1	0	-1
4	-1	0	0	0	-1	-1	-1	-1	+1	-1	-1	-1	-1	-1	+1	-1

1. Problema de la Partición

1 2 3

85

Problema de la Partición

Conociendo la decisión

j:0

i:4 bienes: 4 5 6 7 8 €

4-4=0

i

hayReparto(4,0)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-1	0	0	0	+1	0	0	0	0	0	0	0	0	0	0	0
1	-1	0	0	0	-1	+1	0	0	0	+1	0	0	0	0	0	0
2	-1	0	0	0	-1	-1	+1	0	0	-1	+1	+1	0	0	0	+1
3	-1	0	0	0	-1	-1	-1	+1	0	-1	-1	-1	+1	+1	0	-1
4	-1	0	0	0	-1	-1	-1	-1	+1	-1	-1	-1	-1	-1	+1	-1

1. Problema de la Partición

1 2 3

86

Problema de la Partición

Conociendo la decisión

bienes:

4	5	6	7	8

€

hayReparto(4,0)

j \ i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-1	0	0	0	+1	0	0	0	0	0	0	0	0	0	0	0
1	-1	0	0	0	-1	+1	0	0	0	+1	0	0	0	0	0	0
2	-1	0	0	0	-1	-1	+1	0	0	-1	+1	+1	0	0	0	+1
3	-1	0	0	0	-1	-1	-1	+1	0	-1	-1	-1	+1	+1	0	-1
4	-1	0	0	0	-1	-1	-1	-1	+1	-1	-1	-1	-1	-1	+1	-1

87

Problema de la Partición

Relación de recurrencia

Actividad 15.6. Modifica el algoritmo para que se determine a quién se ha asignado cada bien.

```
boolean[] hayRepartoEquitativo (int[] bienes){
  int objetivo=sumaBienes(bienes)/2;
  int[][] hayReparto = new int[objetivo+1][bienes.length];
  hayReparto[0][0]=-1;
  for (int i=1;i<=objetivo; i++)
       if (bienes[0]==i) hayReparto[i][0]=1; else hayReparto[i][0]=0;
  for (int j=1;j<=bienes.length; j++) {
    for (int i=0;i<=objetivo; i++)
      if (hayReparto[i][j-1]!=0)
         hayReparto[i][j]=-1;
      else if (bienes[j]<=i && hayReparto[i-bienes[j]][j-1]!=0)
         hayReparto[i][j]=+1;
      else
         hayReparto[i][j]=0;
  return obtenerReparto(bienes, hayReparto);
}
```


88

Problema de la Partición

Relación de recurrencia

Actividad 15.6. **Modifica el algoritmo para que se determine a quién se ha asignado cada bien.**

```
boolean[] hayRepartoEquitativo (int[] bienes){
  int objetivo=sumaBienes(bienes)/2;
  int[][] hayReparto = new int[objetivo+1][bienes.length];
  hayReparto[0][0]=-1;
```

```
boolean[] obtenerReparto (int[] bienes, int[][] hayReparto){
  boolean[] reparto = new boolean[bienes.length];
  int i=hayReparto.length-1;
  for (int j=bienes.length-1;j>=0; j--) {
     reparto[j]=(hayReparto[i][j]==1);
     if (reparto[j]) i=i-bienes[j];
  }
  return reparto;
}
```

```
return obtenerReparto(bienes, hayReparto);
}
```

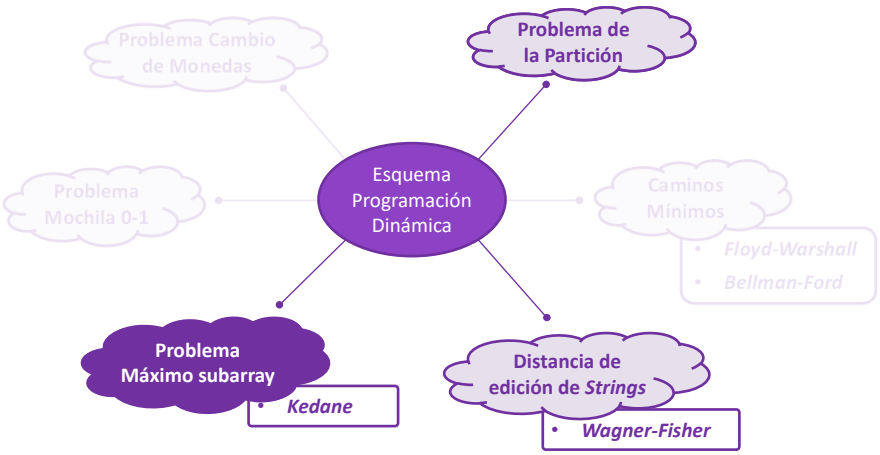
1. Problema de la Partición

1 — 2 — 3

89

Problema del Máximo Subarray

Algunos problemas planteados



2. Problema del Máximo Subarray

1 — 2 — 3

90

Problema del Máximo Subarray

Algunos problemas planteados

Enunciado

Dado un *array*:

-2	12	3	4	-21	8	12	5	-6	1

Encontrar la suma máxima de cualquiera de sus *subarrays*

91

Problema del Máximo Subarray

Enunciado del Problema

Enunciado

Dado un *array*:

-2	**12**	**3**	**4**	-21	8	12	5	-6	1

19

Encontrar la suma máxima de cualquiera de sus *subarrays*

92

Problema del Máximo Subarray

Enunciado del Problema

Enunciado

Dado un *array*:

-2	**12**	**3**	**4**	**-21**	**8**	12	5	-6	1

6

Encontrar la suma máxima de cualquiera de sus *subarrays*

2. Problema del Máximo Subarray

1 — 2 — 3

93

Problema del Máximo Subarray

Enunciado del Problema

Enunciado

Dado un *array*:

-2	12	3	4	**-21**	**8**	**12**	5	-6	1

-1

Encontrar la suma máxima de cualquiera de sus *subarrays*

2. Problema del Máximo Subarray

1 — 2 — 3

94

Problema del Máximo Subarray

Enunciado del Problema

Enunciado

Dado un *array*:

-2	12	3	4	-21	8	12	5	-6	1

25

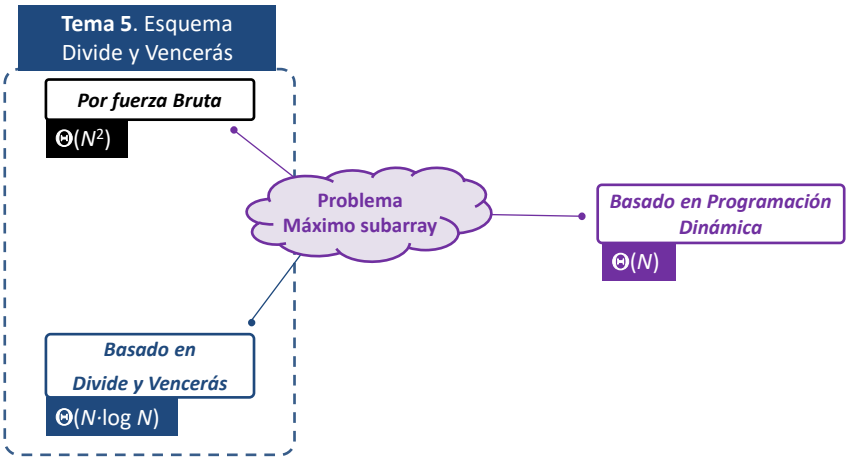
Encontrar la suma máxima de cualquiera de sus *subarrays*

Solución: **25**

95

Problema del Máximo Subarray

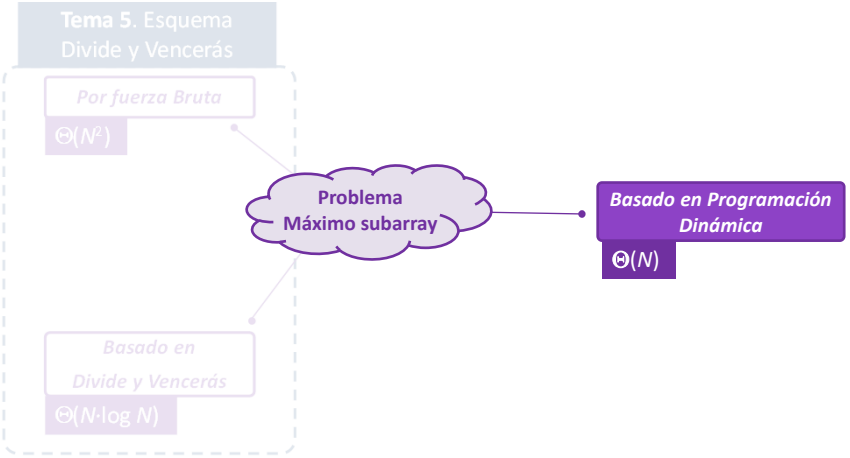
Algoritmos de Resolución



96

Problema del Máximo Subarray

Algoritmos de Resolución



97

Problema del Máximo Subarray

Información Almacenada

i

-2	12	3	4	-21	8	12	5	-6	1

`Max_Total(i)`: Máximo valor del subarray de los elementos `0,…i`

`max_End(i)`: Máximo valor de los subarray que terminan en `i`

98

Problema del Máximo Subarray

Información Almacenada

i=0

-2	12	3	4	-21	8	12	5	-6	1

max_Total(0):-2

max_End(0):-2

99

Problema del Máximo Subarray

Información Almacenada

i=1

-2	12	3	4	-21	8	12	5	-6	1

max_Total(1):12

max_End(1):12

100

Problema del Máximo Subarray

Información Almacenada

i=2

-2	12	3	4	-21	8	12	5	-6	1

max_Total(2):15

max_End(2):15

101

Problema del Máximo Subarray

Información Almacenada

i=3

-2	12	3	4	-21	8	12	5	-6	1

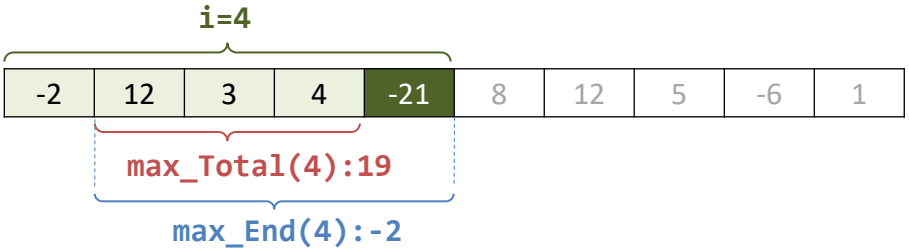
max_Total(3):19

max_End(3):19

102

Problema del Máximo Subarray

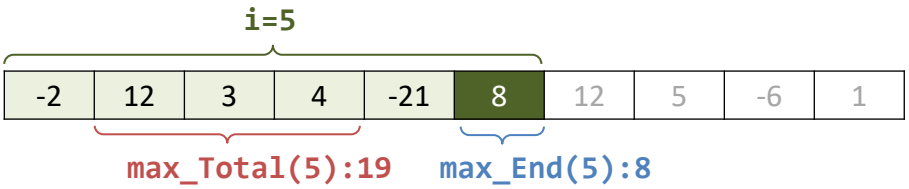
Información Almacenada



103

Problema del Máximo Subarray

Información Almacenada



104

Problema del Máximo Subarray

Información Almacenada

i=6

-2	12	3	4	-21	8	12	5	-6	1

max_Total(6):20
max_End(6):20

105

Problema del Máximo Subarray

Información Almacenada

i=7

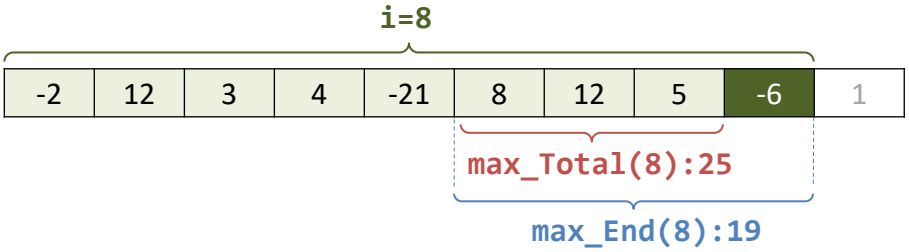
-2	12	3	4	-21	8	12	5	-6	1

max_Total(7):25
max_End(7):25

106

Problema del Máximo Subarray

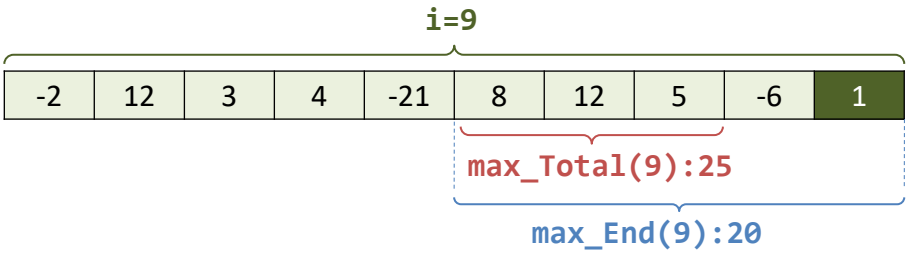
Información Almacenada



107

Problema del Máximo Subarray

Información Almacenada



108

Problema del Máximo Subarray

Enunciado del Problema

-2	12	3	4	-21	**8**	**12**	**5**	-6	1

max_Total:25

Actividad 15.7. Encuentra una ecuación de recurrencia para definir max_End(i) y max_Total(i).

109

Problema del Máximo Subarray

Relación de Recurrencia

i=0

-2	12	3	4	-21	8	12	5	-6	1

max_End(**0**)=vector[0]

max_Total(**0**)=vector[0]

Actividad 15.7. Encuentra una ecuación de recurrencia para definir max_End(i) y max_Total(i).

110

Problema del Máximo Subarray

Enunciado del Problema

i

-2	12	3	4	-21	8	12	5	-6	1

max_End(0)=vector[0]

max_Total(0)=vector[0]

max_End(i)= max{vector[i], *max_End*(i-1)+vector[i]}

max_Total(i)= max{*max_End*(i), *max_Total*(i-1)}

Actividad 15.7. **Encuentra una ecuación de recurrencia para definir max_End(i) y max_Total(i).**

111

Problema del Máximo Subarray

Enunciado del Problema

Actividad 15.8. **Implementa el algoritmo que determina el valor del máximo subarray usando programación dinámica.**

max_End(0)=vector[0]

max_Total(0)=vector[0]

max_End(i)= max{vector[i], *max_End*(i-1)+vector[i]}

max_Total(i)= max{*max_End*(i), *max_Total*(i-1)}

-2	12	3	4	-21	8	12	5	-6	1

```
i=0

max_End:   -2
max_Total: -2
```


112

Problema del Máximo Subarray

Enunciado del Problema

Actividad 15.8. **Implementa el algoritmo que determina el valor del máximo subarray usando programación dinámica.**

max_End(0)=vector[0]
max_Total(0)=vector[0]

max_End(i)= max{vector[i], *max_End*(i-1)+vector[i]}
max_Total(i)= max{*max_End*(i), *max_Total*(i-1)}

-2	12	3	4	-21	8	12	5	-6	1

```
i=1

max_End:    12
max_Total:  12
```


113

Problema del Máximo Subarray

Enunciado del Problema

Actividad 15.8. **Implementa el algoritmo que determina el valor del máximo subarray usando programación dinámica.**

max_End(0)=vector[0]
max_Total(0)=vector[0]

max_End(i)= max{vector[i], *max_End*(i-1)+vector[i]}
max_Total(i)= max{*max_End*(i), *max_Total*(i-1)}

-2	12	3	4	-21	8	12	5	-6	1

```
i=2

max_End:    15
max_Total:  15
```


114

Problema del Máximo Subarray

Enunciado del Problema

Actividad 15.8. **Implementa el algoritmo que determina el valor del máximo subarray usando programación dinámica.**

max_End(0)=vector[0]

max_Total(0)=vector[0]

max_End(i)= max{vector[i], *max_End*(i-1)+vector[i]}

max_Total(i)= max{*max_End*(i), *max_Total*(i-1)}

-2	12	3	4	-21	8	12	5	-6	1

```
i=3

max_End:    19
max_Total:  19
```


115

Problema del Máximo Subarray

Enunciado del Problema

Actividad 15.8. **Implementa el algoritmo que determina el valor del máximo subarray usando programación dinámica.**

max_End(0)=vector[0]

max_Total(0)=vector[0]

max_End(i)= max{vector[i], *max_End*(i-1)+vector[i]}

max_Total(i)= max{*max_End*(i), *max_Total*(i-1)}

-2	12	3	4	-21	8	12	5	-6	1

```
i=4

max_End:    -2
max_Total:  19
```


116

Problema del Máximo Subarray

Enunciado del Problema

Actividad 15.8. **Implementa el algoritmo que determina el valor del máximo subarray usando programación dinámica.**

max_End(0)=vector[0]
max_Total(0)=vector[0]

max_End(i)= max{vector[i], *max_End*(i-1)+vector[i]}
max_Total(i)= max{*max_End*(i), *max_Total*(i-1)}

-2	12	3	4	-21	8	12	5	-6	1

```
i=5

max_End:     8
max_Total: 19
```


117

Problema del Máximo Subarray

Enunciado del Problema

Actividad 15.8. **Implementa el algoritmo que determina el valor del máximo subarray usando programación dinámica.**

max_End(0)=vector[0]
max_Total(0)=vector[0]

max_End(i)= max{vector[i], *max_End*(i-1)+vector[i]}
max_Total(i)= max{*max_End*(i), *max_Total*(i-1)}

-2	12	3	4	-21	8	12	5	-6	1

```
i=6

max_End:    20
max_Total: 20
```


118

Problema del Máximo Subarray

Enunciado del Problema

Actividad 15.8. Implementa el algoritmo que determina el valor del máximo subarray usando programación dinámica.

max_End(0)=vector[0]
max_Total(0)=vector[0]

max_End(i)= max{vector[i], *max_End*(i-1)+vector[i]}
max_Total(i)= max{*max_End*(i), *max_Total*(i-1)}

-2	12	3	4	-21	8	12	5	-6	1

```
i=7

max_End:    25
max_Total:  25
```


119

Problema del Máximo Subarray

Enunciado del Problema

Actividad 15.8. Implementa el algoritmo que determina el valor del máximo subarray usando programación dinámica.

max_End(0)=vector[0]
max_Total(0)=vector[0]

max_End(i)= max{vector[i], *max_End*(i-1)+vector[i]}
max_Total(i)= max{*max_End*(i), *max_Total*(i-1)}

-2	12	3	4	-21	8	12	5	-6	1

```
i=8

max_End:    19
max_Total:  25
```


120

Problema del Máximo Subarray

Enunciado del Problema

Actividad 15.8. Implementa el algoritmo que determina el valor del máximo subarray usando programación dinámica.

max_End(0)=vector[0]

max_Total(0)=vector[0]

max_End(i)= max{vector[i], *max_End*(i-1)+vector[i]}

max_Total(i)= max{*max_End*(i), *max_Total*(i-1)}

-2	12	3	4	-21	8	12	5	-6	1

```
i=9
max_End:   20
max_Total: 25
```


121

Problema del Máximo Subarray

Enunciado del Problema

Actividad 15.8. Implementa el algoritmo que determina el valor del máximo subarray usando programación dinámica.

max_End(0)=vector[0]

max_Total(0)=vector[0]

max_End(i)= max{vector[i], *max_End*(i-1)+vector[i]}

max_Total(i)= max{*max_End*(i), *max_Total*(i-1)}

```
int maxSubarray (int[] vector){
  int max_end=vector[0];
  int max_total=vector[0];
  for (int i=1;i<vector.length; i++) {
     max_end=Math.max(max_end+vector[i], vector[i]);
     max_total=Math.max(max_end, max_total);
  }
  return max_total;
}
```


122

Problema del Máximo Subarray

Enunciado del Problema

Actividad 15.8. Implementa el algoritmo que determina el valor del máximo subarray usando programación dinámica.

max_End(0)=vector[0]
max_Total(0)=vector[0]

max_End(i)= max{vector[i], *max_End*(i-1)+vector[i]}
max_Total(i)= max{*max_End*(i), *max_Total*(i-1)}

```
int maxSubarray (int[] vector){
  int max_end=vector[0];
  int max_total=vector[0];
  for (int i=1;i<vector.length; i++) {
     max_end=Math.max(max_end+vector[i], vector[i]);
     max_total=Math.max(max_end, max_total);
  }
  return max_total;
}
```

2. Problema del Máximo Subarray

1 — 2 — 3

123

Problema del Máximo Subarray

Enunciado del Problema

Actividad 15.8. Implementa el algoritmo que determina el valor del máximo subarray usando programación dinámica.

max_End(0)=vector[0]
max_Total(0)=vector[0]

max_End(i)= max{vector[i], *max_End*(i-1)+vector[i]}
max_Total(i)= max{*max_End*(i), *max_Total*(i-1)}

```
int maxSubarray (int[] vector){
  int max_end=vector[0];
  int max_total=vector[0];
  for (int i=1;i<vector.length; i++) {
     max_end=Math.max(max_end+vector[i], vector[i]);
     max_total=Math.max(max_end, max_total);
  }
  return max_total;
}
```

2. Problema del Máximo Subarray

1 — 2 — 3

124

Problema del Máximo Subarray

Enunciado del Problema

Actividad 15.9. Calcula la complejidad del algoritmo en tiempo y memoria en función del tamaño del array, *N*.

max_End(0)=vector[0]

max_Total(0)=vector[0]

max_End(i)= max{vector[i], *max_End*(i-1)+vector[i]}

max_Total(i)= max{*max_End*(i), *max_Total*(i-1)}

```
int maxSubarray (int[] vector){
  int max_end=vector[0];
  int max_total=vector[0];
  for (int i=1;i<vector.length; i++) {
     max_end=Math.max(max_end+vector[i], vector[i]);
     max_total=Math.max(max_end, max_total);
  }
  return max_total;
}
```

En memoria: Θ(1)
En tiempo: Θ(*N*)

2. Problema del Máximo Subarray

1 — 2 — 3

125

Problema del Máximo Subarray

Enunciado del Problema

Actividad 15.10. Modifica el algoritmo para que devuelva el subarray máximo usando programación dinámica.

max_End(0)=vector[0]

max_Total(0)=vector[0]

max_End(i)= max{vector[i], *max_End*(i-1)+vector[i]}

max_Total(i)= max{*max_End*(i), *max_Total*(i-1)}

```
int maxSubarray (int[] vector, int[] subarray){
 int max_end=vector[0]; int i0=0; int iN=0;
 int max_total=vector[0]; int j0=0; int jN=0;
 for (int i=1;i<vector.length; i++) {
    if (max_end>0){
       max_end=max_end+vector[i]; iN=i;}
    else {
       max_end=vector[i]; i0=i; iN=i;}
    if (max_total<max_end){
       max_total=max_end; j0=i0; jN=iN;}
 }
 subarray[0]=j0; subarray[1]=jN;
 return max_total;
}
```

126

Problema del Máximo Subarray

Enunciado del Problema

Actividad 15.10. Modifica el algoritmo para que devuelva el subarray máximo usando programación dinámica.

max_End(0)=vector[0]

max_Total(0)=vector[0]

max_End(i)= max{vector[i], *max_End*(i-1)+vector[i]}

max_Total(i)= max{*max_End*(i), *max_Total*(i-1)}

```
int maxSubarray (int[] vector, int[] subarray){
 int max_end=vector[0]; int i0=0; int iN=0;
 int max_total=vector[0]; int j0=0; int jN=0;
 for (int i=1;i<vector.length; i++) {
    if (max_end>0){
       max_end=max_end+vector[i]; iN=i;}
    else {
       max_end=vector[i]; i0=i; iN=i;}
    if (max_total<max_end){
       max_total=max_end; j0=i0; jN=iN;}
 }
 subarray[0]=j0; subarray[1]=jN;
 return max_total;
}
```

127

Problema del Máximo Subarray

Enunciado del Problema

Actividad 15.10. Modifica el algoritmo para que devuelva el subarray máximo usando programación dinámica.

max_End(0)=vector[0]

max_Total(0)=vector[0]

max_End(i)= max{vector[i], *max_End*(i-1)+vector[i]}

max_Total(i)= max{*max_End*(i), *max_Total*(i-1)}

```
int maxSubarray (int[] vector, int[] subarray){
 int max_end=vector[0]; int i0=0; int iN=0;
 int max_total=vector[0]; int j0=0; int jN=0;
 for (int i=1;i<vector.length; i++) {
    if (max_end>0){
       max_end=max_end+vector[i]; iN=i;}
    else {
       max_end=vector[i]; i0=i; iN=i;}
    if (max_total<max_end){
       max_total=max_end; j0=i0; jN=iN;}
 }
 subarray[0]=j0; subarray[1]=jN;
 return max_total;
}
```

128

Problema del Máximo Subarray

Enunciado del Problema

Actividad 15.10. Modifica el algoritmo para que devuelva el subarray máximo usando programación dinámica.

max_End(0)=vector[0]

max_Total(0)=vector[0]

max_End(i)= max{vector[i], *max_End*(i-1)+vector[i]}

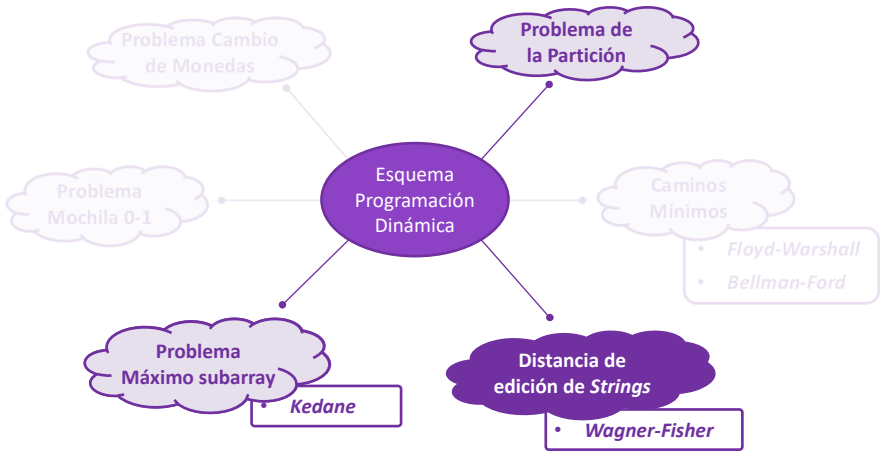
max_Total(i)= max{*max_End*(i), *max_Total*(i-1)}

```
int maxSubarray (int[] vector, int[] subarray){
 int max_end=vector[0]; int i0=0; int iN=0;
 int max_total=vector[0]; int j0=0; int jN=0;
 for (int i=1;i<vector.length; i++) {
    if (max_end>0){
       max_end=max_end+vector[i]; iN=i;}
    else {
       max_end=vector[i]; i0=i; iN=i;}
    if (max_total<max_end){
       max_total=max_end; j0=i0; jN=iN;}
 }
 subarray[0]=j0; subarray[1]=jN;
 return max_total;
}
```

129

Distancia de edición de Strings

Algunos problemas planteados



3. Distancia de Strings

1 2 3

130

Distancia de edición de Strings

Enunciado del Problema

Dados dos strings:

s_1: i n f l i g i r → i n f r i g i r → i n f r i n g i r

Sustituir l por r

Añadir n

s_2: i n f r i n g i r

d(“*infligir*”, “*infringir*”) = **2**

Distancia de Levenshtein

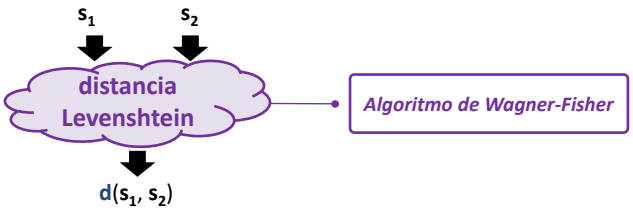
d(s_1, s_2):Número mínimo de **cambios de edición** para transformar una palabra en otra

- **Borrar** un carácter
- **Insertar** un carácter
- **Sustituir** un carácter por otro

131

Distancia de edición de Strings

Enunciado del Problema



132

Distancia de edición de Strings

Ecuaciones recursivas

lev(*i*, *j*): mínimo número de cambios que hay que hacer en:

- Los *i* primeros caracteres de s_1.
- Los *j* primeros caracteres de s_2.

133

Distancia de edición de Strings

Ecuaciones recursivas

lev(4, 3): mínimo número de cambios que hay que hacer en:

- Los **4** primeros caracteres de s_1.
- Los **3** primeros caracteres de s_2.

134

Distancia de edición de Strings

Ecuaciones recursivas

lev(2, 3): mínimo número de cambios que hay que hacer en:

- Los **2** primeros caracteres de s_1.
- Los **3** primeros caracteres de s_2.

135

Distancia de edición de Strings

Ecuaciones recursivas

lev(0, 3): mínimo número de cambios que hay que hacer en:

- Ningún carácter de s_1.
- Los **3** primeros caracteres de s_2.

136

Distancia de edición de Strings

Ecuaciones recursivas

lev(8, 9): mínimo número de cambios que hay que hacer en:

- Los **8** primeros caracteres de s_1.
- Los **9** primeros caracteres de s_2.

Distancia de Levenshtein

137

Distancia de edición de Strings

Ecuaciones recursivas

lev(*i*, *j*): mínimo número de cambios que hay que hacer en:

- Los *i* primeros caracteres de s_1.
- Los *j* primeros caracteres de s_2.

Caso Base:

lev(0, *j*) = *j* lev(*i*, 0) = *i*

138

Distancia de edición de Strings

Ecuaciones recursivas

lev(*i*, *j*): mínimo número de cambios que hay que hacer en:

- Los *i* primeros caracteres de s_1.
- Los *j* primeros caracteres de s_2.

Caso Recursivo:

$$lev(i,j)=\begin{cases} lev(i-1,j-1) & si\ s_1[i]=s_2[j] \\ 1+\min\{lev(i,j-1),\ lev(i-1,j-1),\ lev(i-1,j)\} & en\ otro\ caso \end{cases}$$

139

Distancia de edición de Strings

Ecuaciones recursivas

lev(*i*, *j*): mínimo número de cambios que hay que hacer en:

- Los *i* primeros caracteres de s_1.
- Los *j* primeros caracteres de s_2.

Caso Recursivo:

No se hace nada si el carácter *i-ésimo* de s_1 es el mismo que el carácter *j*-ésimo de s_2

$$lev(i,j)=\begin{cases} \boxed{lev(i-1,j-1)} & si\ s_1[i]=s_2[j] \\ 1+\min\{lev(i,j-1),\ lev(i-1,j-1),\ lev(i-1,j)\} & en\ otro\ caso \end{cases}$$

140

Distancia de edición de Strings

Ecuaciones recursivas

lev(*i*, *j*): mínimo número de cambios que hay que hacer en:

- Los *i* primeros caracteres de s_1.
- Los *j* primeros caracteres de s_2.

Caso Recursivo:

$$lev(i,j)=\begin{cases} lev(i-1,j-1) & si\ s_1[i]=s_2[j] \\ 1+\min\{\boxed{lev(i,j-1),}\ lev(i-1,j-1),\ lev(i-1,j)\} & en\ otro\ caso \end{cases}$$

Añadir en la posición *i-ésima+1* de s_1 el carácter *j*-ésimo de s_2

141

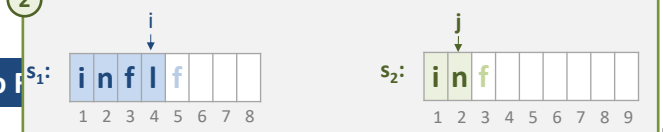
Distancia de edición de Strings

Ecuaciones recursivas

lev(*i*, *j*): mínimo número de cambios que hay que hacer en:

- Los *i* primeros caracteres de s_1.

2



Caso R

$$lev(i,j)=\begin{cases} \text{[illegible]} & \text{[illegible]}\ s_1[i]=s_2[j] \\ 1+\min\{\boxed{lev(i,j-1),}\ lev(i-1,j-1),\ lev(i-1,j)\} & en\ otro\ caso \end{cases}$$

Añadir en la posición *i-ésima+1* de s_1 el carácter *j*-ésimo de s_2

142

Distancia de edición de Strings

Ecuaciones recursivas

lev(*i*, *j*): mínimo número de cambios que hay que hacer en:

- Los *i* primeros caracteres de s_1.
- Los *j* primeros caracteres de s_2.

Caso Recursivo:

$$lev(i,j) = \begin{cases} lev(i-1, j-1) & si\ s_1[i] = s_2[j] \\ 1 + \min\{lev(i, j-1),\ \boxed{lev(i-1, j-1)},\ lev(i-1, j)\} & en\ otro\ caso \end{cases}$$

Sustituir el carácter *i-ésimo* de s_1 por el carácter *j*-ésimo de s_2

143

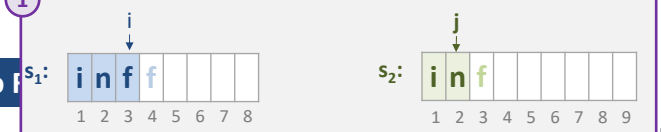
Distancia de edición de Strings

Ecuaciones recursivas

lev(*i*, *j*): mínimo número de cambios que hay que hacer en:

- Los *i* primeros caracteres de s_1.

1



Caso R

$$lev(i,j) = \begin{cases} lev(i-1, j-1) & si\ s_1[i] = s_2[j] \\ 1 + \min\{lev(i, j-1),\ \boxed{lev(i-1, j-1)},\ lev(i-1, j)\} & en\ otro\ caso \end{cases}$$

Sustituir el carácter *i-ésimo* de s_1 por el carácter *j*-ésimo de s_2

3. Distancia de Strings
1 2 3

144

Distancia de edición de Strings

Ecuaciones recursivas

lev(*i*, *j*): mínimo número de cambios que hay que hacer en:

- Los ***i*** primeros caracteres de s_1.
- Los ***j*** primeros caracteres de s_2.

Caso Recursivo:

$$lev(i,j)=\begin{cases} lev(i-1,j-1) & si\ s_1[i]=s_2[j] \\ 1+\min\{lev(i,j-1),\ lev(i-1,j-1),\ \boxed{lev(i-1,j)}\} & en\ otro\ caso \end{cases}$$

Borrar el ***i-ésimo*** carácter de s_1

145

Distancia de edición de Strings

Ecuaciones recursivas

lev(*i*, *j*): mínimo número de cambios que hay que hacer en:

- Los ***i*** primeros caracteres de s_1.

0

Caso R

$$lev(i,j)=\begin{cases} lev(i-1,j-1) & si\ s_1[i]=s_2[j] \\ 1+\min\{lev(i,j-1),\ lev(i-1,j-1),\ \boxed{lev(i-1,j)}\} & en\ otro\ caso \end{cases}$$

Borrar el ***i-ésimo*** carácter de s_1

146

Distancia de edición de Strings

Ecuaciones recursivas

lev(i, 0) = i lev(0, j) = j

		0	1	2	3	4	5	6	7	8	9
			i	n	f	r	i	n	g	i	r
0		0	1	2	3	4	5	6	7	8	9
1	i	1									
2	n	2									
3	f	3									
4	l	4									
5	i	5									
6	g	6									
7	i	7									
8	r	8									

147

Distancia de edición de Strings

Ecuaciones recursivas

$$lev(i,j) = \begin{cases} \boxed{lev(i-1,j-1)} & si\ s_1[i] = s_2[j] \\ 1 + \min\{lev(i,j-1),\ lev(i-1,j-1),\ lev(i-1,j)\} & en\ otro\ caso \end{cases}$$

		0	1	2	3	4	5	6	7	8	9
			i	n	f	r	i	n	g	i	r
0		0	1	2	3	4	5	6	7	8	9
1	i	1	0								
2	n	2									
3	f	3									
4	l	4									
5	i	5									
6	g	6									
7	i	7									
8	r	8									

148

Distancia de edición de Strings

Ecuaciones recursivas

$$lev(i,j) = \begin{cases} lev(i-1, j-1) & si\ s_1[i] = s_2[j] \\ 1 + \min\{lev(i, j-1), lev(i-1, j-1), lev(i-1, j)\} & en\ otro\ caso \end{cases}$$

		0	1	2	3	4	5	6	7	8	9
			i	n	f	r	i	n	g	i	r
0		0	1	2	3	4	5	6	7	8	9
1	i	1	0	1							
2	n	2									
3	f	3									
4	l	4									
5	i	5									
6	g	6									
7	i	7									
8	r	8									

149

Distancia de edición de Strings

Ecuaciones recursivas

$$lev(i,j) = \begin{cases} lev(i-1, j-1) & si\ s_1[i] = s_2[j] \\ 1 + \min\{lev(i, j-1), lev(i-1, j-1), lev(i-1, j)\} & en\ otro\ caso \end{cases}$$

		0	1	2	3	4	5	6	7	8	9
			i	n	f	r	i	n	g	i	r
0		0	1	2	3	4	5	6	7	8	9
1	i	1	0	1	2						
2	n	2									
3	f	3									
4	l	4									
5	i	5									
6	g	6									
7	i	7									
8	r	8									

150

Distancia de edición de Strings

Ecuaciones recursivas

$$lev(i,j) = \begin{cases} lev(i-1,j-1) & si\ s_1[i] = s_2[j] \\ 1 + \min\{\boxed{lev(i,j-1)}, \boxed{lev(i-1,j-1)}, \boxed{lev(i-1,j)}\} & en\ otro\ caso \end{cases}$$

		0	1	2	3	4	5	6	7	8	9
			i	n	f	r	i	n	g	i	r
0		0	1	2	3	4	5	6	7	8	9
1	i	1	0	1	2	3					
2	n	2									
3	f	3									
4	l	4									
5	i	5									
6	g	6									
7	i	7									
8	r	8									

151

Distancia de edición de Strings

Ecuaciones recursivas

$$lev(i,j) = \begin{cases} lev(i-1,j-1) & si\ s_1[i] = s_2[j] \\ 1 + \min\{\boxed{lev(i,j-1)}, \boxed{lev(i-1,j-1)}, \boxed{lev(i-1,j)}\} & en\ otro\ caso \end{cases}$$

		0	1	2	3	4	5	6	7	8	9
			i	n	f	r	i	n	g	i	r
0		0	1	2	3	4	5	6	7	8	9
1	i	1	0	1	2	3	4	5	6	7	8
2	n	2									
3	f	3									
4	l	4									
5	i	5									
6	g	6									
7	i	7									
8	r	8									

152

Distancia de edición de Strings

Ecuaciones recursivas

$$lev(i,j) = \begin{cases} lev(i-1,j-1) & si\ s_1[i] = s_2[j] \\ 1 + \min\{\boxed{lev(i,j-1)}, \boxed{lev(i-1,j-1)}, \boxed{lev(i-1,j)}\} & en\ otro\ caso \end{cases}$$

		0	1	2	3	4	5	6	7	8	9
			i	n	f	r	i	n	g	i	r
0		0	1	2	3	4	5	6	7	8	9
1	i	1	0	1	2	3	4	5	6	7	8
2	n	2	1								
3	f	3									
4	l	4									
5	i	5									
6	g	6									
7	i	7									
8	r	8									

153

Distancia de edición de Strings

Ecuaciones recursivas

$$lev(i,j) = \begin{cases} \boxed{lev(i-1,j-1)} & si\ s_1[i] = s_2[j] \\ 1 + \min\{lev(i,j-1),\ lev(i-1,j-1),\ lev(i-1,j)\} & en\ otro\ caso \end{cases}$$

		0	1	2	3	4	5	6	7	8	9
			i	n	f	r	i	n	g	i	r
0		0	1	2	3	4	5	6	7	8	9
1	i	1	0	1	2	3	4	5	6	7	8
2	n	2	1	0							
3	f	3									
4	l	4									
5	i	5									
6	g	6									
7	i	7									
8	r	8									

154

Distancia de edición de Strings

Ecuaciones recursivas

$$lev(i,j) = \begin{cases} lev(i-1, j-1) & si\ s_1[i] = s_2[j] \\ 1 + \min\{\boxed{lev(i, j-1)}, \boxed{lev(i-1, j-1)}, \boxed{lev(i-1, j)}\} & en\ otro\ caso \end{cases}$$

		0	1	2	3	4	5	6	7	8	9
			i	n	f	r	i	n	g	i	r
0		0	1	2	3	4	5	6	7	8	9
1	i	1	0	1	2	3	4	5	6	7	8
2	n	2	1	0	1	2	3	4	5	6	7
3	f	3									
4	l	4									
5	i	5									
6	g	6									
7	i	7									
8	r	8									

3. Distancia de Strings

1 — 2 — 3

155

Distancia de edición de Strings

Ecuaciones recursivas

$$lev(i,j) = \begin{cases} lev(i-1, j-1) & si\ s_1[i] = s_2[j] \\ 1 + \min\{\boxed{lev(i, j-1)}, \boxed{lev(i-1, j-1)}, \boxed{lev(i-1, j)}\} & en\ otro\ caso \end{cases}$$

		0	1	2	3	4	5	6	7	8	9
			i	n	f	r	i	n	g	i	r
0		0	1	2	3	4	5	6	7	8	9
1	i	1	0	1	2	3	4	5	6	7	8
2	n	2	1	0	1	2	3	4	5	6	7
3	f	3	2	1	0	1	2	3	4	5	6
4	l	4									
5	i	5									
6	g	6									
7	i	7									
8	r	8									

3. Distancia de Strings

1 — 2 — 3

156

Distancia de edición de Strings

Ecuaciones recursivas

$$lev(i,j) = \begin{cases} lev(i-1,j-1) & si\ s_1[i] = s_2[j] \\ 1 + \min\{\boxed{lev(i,j-1)}, \boxed{lev(i-1,j-1)}, \boxed{lev(i-1,j)}\} & en\ otro\ caso \end{cases}$$

		0	1	2	3	4	5	6	7	8	9
			i	n	f	r	i	n	g	i	r
0		0	1	2	3	4	5	6	7	8	9
1	i	1	0	1	2	3	4	5	6	7	8
2	n	2	1	0	1	2	3	4	5	6	7
3	f	3	2	1	0	1	2	3	4	5	6
4	l	4	3	2	1	1					
5	i	5									
6	g	6									
7	i	7									
8	r	8									

157

Distancia de edición de Strings

Ecuaciones recursivas

$$lev(i,j) = \begin{cases} lev(i-1,j-1) & si\ s_1[i] = s_2[j] \\ 1 + \min\{\boxed{lev(i,j-1)}, \boxed{lev(i-1,j-1)}, \boxed{lev(i-1,j)}\} & en\ otro\ caso \end{cases}$$

		0	1	2	3	4	5	6	7	8	9
			i	n	f	r	i	n	g	i	r
0		0	1	2	3	4	5	6	7	8	9
1	i	1	0	1	2	3	4	5	6	7	8
2	n	2	1	0	1	2	3	4	5	6	7
3	f	3	2	1	0	1	2	3	4	5	6
4	l	4	3	2	1	1	2	3	4	5	6
5	i	5									
6	g	6									
7	i	7									
8	r	8									

158

Distancia de edición de Strings

Ecuaciones recursivas

$$lev(i,j) = \begin{cases} lev(i-1,j-1) & si\ s_1[i] = s_2[j] \\ 1 + \min\{\boxed{lev(i,j-1)}, \boxed{lev(i-1,j-1)}, \boxed{lev(i-1,j)}\} & en\ otro\ caso \end{cases}$$

		0	1	2	3	4	5	6	7	8	9
			i	n	f	r	i	n	g	i	r
0		0	1	2	3	4	5	6	7	8	9
1	i	1	0	1	2	3	4	5	6	7	8
2	n	2	1	0	1	2	3	4	5	6	7
3	f	3	2	1	0	1	2	3	4	5	6
4	l	4	3	2	1	1	2	3	4	5	6
5	i	5	4	3	2	2					
6	g	6									
7	i	7									
8	r	8									

159

Distancia de edición de Strings

Ecuaciones recursivas

$$lev(i,j) = \begin{cases} \boxed{lev(i-1,j-1)} & si\ s_1[i] = s_2[j] \\ 1 + \min\{lev(i,j-1),\ lev(i-1,j-1),\ lev(i-1,j)\} & en\ otro\ caso \end{cases}$$

		0	1	2	3	4	5	6	7	8	9
			i	n	f	r	i	n	g	i	r
0		0	1	2	3	4	5	6	7	8	9
1	i	1	0	1	2	3	4	5	6	7	8
2	n	2	1	0	1	2	3	4	5	6	7
3	f	3	2	1	0	1	2	3	4	5	6
4	l	4	3	2	1	1	2	3	4	5	6
5	i	5	4	3	2	2	1				
6	g	6									
7	i	7									
8	r	8									

160

Distancia de edición de Strings

Ecuaciones recursivas

$$lev(i,j) = \begin{cases} lev(i-1,j-1) & si\ s_1[i] = s_2[j] \\ 1 + \min\{\boxed{lev(i,j-1)}, \boxed{lev(i-1,j-1)}, \boxed{lev(i-1,j)}\} & en\ otro\ caso \end{cases}$$

		0	1	2	3	4	5	6	7	8	9
			i	n	f	r	i	n	g	i	r
0		0	1	2	3	4	5	6	7	8	9
1	i	1	0	1	2	3	4	5	6	7	8
2	n	2	1	0	1	2	3	4	5	6	7
3	f	3	2	1	0	1	2	3	4	5	6
4	l	4	3	2	1	1	2	3	4	5	6
5	i	5	4	3	2	2	1	2	3	4	5
6	g	6									
7	i	7									
8	r	8									

161

Distancia de edición de Strings

Ecuaciones recursivas

$$lev(i,j) = \begin{cases} lev(i-1,j-1) & si\ s_1[i] = s_2[j] \\ 1 + \min\{\boxed{lev(i,j-1)}, \boxed{lev(i-1,j-1)}, \boxed{lev(i-1,j)}\} & en\ otro\ caso \end{cases}$$

		0	1	2	3	4	5	6	7	8	9
			i	n	f	r	i	n	g	i	r
0		0	1	2	3	4	5	6	7	8	9
1	i	1	0	1	2	3	4	5	6	7	8
2	n	2	1	0	1	2	3	4	5	6	7
3	f	3	2	1	0	1	2	3	4	5	6
4	l	4	3	2	1	1	2	3	4	5	6
5	i	5	4	3	2	2	1	2	3	4	5
6	g	6	5	4	3	3	2	2	2	3	4
7	i	7									
8	r	8									

162

Distancia de edición de Strings

Ecuaciones recursivas

$$lev(i,j) = \begin{cases} \boxed{lev(i-1,j-1)} & si\ s_1[i] = s_2[j] \\ 1 + \min\{lev(i,j-1),\ lev(i-1,j-1),\ lev(i-1,j)\} & en\ otro\ caso \end{cases}$$

		0	1	2	3	4	5	6	7	8	9
			i	n	f	r	i	n	g	i	r
0		0	1	2	3	4	5	6	7	8	9
1	i	1	0	1	2	3	4	5	6	7	8
2	n	2	1	0	1	2	3	4	5	6	7
3	f	3	2	1	0	1	2	3	4	5	6
4	l	4	3	2	1	1	2	3	4	5	6
5	i	5	4	3	2	2	1	2	3	4	5
6	g	6	5	4	3	3	2	2	2	3	4
7	i	7	6	5	4	4	3	3	3	2	
8	r	8									

163

Distancia de edición de Strings

Ecuaciones recursivas

$$lev(i,j) = \begin{cases} \boxed{lev(i-1,j-1)} & si\ s_1[i] = s_2[j] \\ 1 + \min\{lev(i,j-1),\ lev(i-1,j-1),\ lev(i-1,j)\} & en\ otro\ caso \end{cases}$$

		0	1	2	3	4	5	6	7	8	9
			i	n	f	r	i	n	g	i	r
0		0	1	2	3	4	5	6	7	8	9
1	i	1	0	1	2	3	4	5	6	7	8
2	n	2	1	0	1	2	3	4	5	6	7
3	f	3	2	1	0	1	2	3	4	5	6
4	l	4	3	2	1	1	2	3	4	5	6
5	i	5	4	3	2	2	1	2	3	4	5
6	g	6	5	4	3	3	2	2	2	3	4
7	i	7	6	5	4	4	3	3	3	2	3
8	r	8	7	6	5	4	4	4	4	3	2

164

Distancia de edición de Strings

Ecuaciones recursivas

$$lev(i,j) = \begin{cases} lev(i-1,j-1) & si\ s_1[i] = s_2[j] \\ 1 + \min\{lev(i,j-1),\ lev(i-1,j-1),\ lev(i-1,j)\} & en\ otro\ caso \end{cases}$$

		0	1	2	3	4	5	6	7	8	9
			i	n	f	r	i	n	g	i	r
0		0	1	2	3	4	5	6	7	8	9
1	i	1	0	1	2	3	4	5	6	7	8
2	n	2	1	0	1	2	3	4	5	6	7
3	f	3	2	1	0	1	2	3	4	5	6
4	l	4	3	2	1	1	2	3	4	5	6
5	i	5	4	3	2	2	1	2	3	4	5
6	g	6	5	4	3	3	2	2	2	3	4
7	i	7	6	5	4	4	3	3	3	2	3
8	r	8	7	6	5	4	4	4	4	3	**2**

d(“*infligir*”, “*infringir*”) = **2**

165

Problema de la Partición

Relación de recurrencia

Actividad 15.11. Implementa el algoritmo que determina la distancia de Levenshtein.

```
int distanciaLevenshtein (String s1, String s2){
  int long1= s1.length(); int long2= s2.length();
  int[][] lev = new int[long1+1][long2+1];
  for (int i=0; i<=long1; i++)
     lev[i][0]=i;
  for (int j=0; j<=long2; j++)
     lev[0][j]=j;
  for (int i=1; i<=long1; i++)
    for (int j=1; j<=long2; j++)
      if (s1.charAt(i-1)==s2.charAt(j-1))
         lev[i][j]=lev[i-1][j-1];
      else
         lev[i][j]= 1 + Math.min(lev[i-1][j-1],
                                 Math.min(lev[i-1][j],lev[i][j-1]));
  return lev[long1][long2];
}
```


166

Distancia de edición de Strings

Relación de recurrencia

Actividad 15.11. **Implementa el algoritmo que determina la distancia de Levenshtein.**

```
int distanciaLevenshtein (String s1, String s2){
  int long1= s1.length(); int long2= s2.length();
  int[][] lev = new int[long1+1][long2+1];
  for (int i=0; i<=long1; i++)
     lev[i][0]=i;
  for (int j=0; j<=long2; j++)
     lev[0][j]=j;
  for (int i=1; i<=long1; i++)
    for (int j=1; j<=long2; j++)
      if (s1.charAt(i-1)==s2.charAt(j-1))
         lev[i][j]=lev[i-1][j-1];
      else
         lev[i][j]= 1 + Math.min(lev[i-1][j-1],
                                 Math.min(lev[i-1][j],lev[i][j-1]));
  return lev[long1][long2];
}
```

Caso Base:

lev(0, *j*) = *j* lev(*i*, 0) = *i*

167

Distancia de edición de Strings

Relación de recurrencia

Actividad 15.11. **Implementa el algoritmo que determina la distancia de Levenshtein.**

```
int distanciaLevenshtein (String s1, String s2){
  int long1= s1.length(); int long2= s2.length();
  int[][] lev = new int[long1+1][long2+1];
  for (int i=0; i<=long1; i++)
```

Caso Recursivo:

$$lev(i,j) = \begin{cases} lev(i-1, j-1) & si\ s_1[i] = s_2[j] \\ 1 + \min\{lev(i, j-1),\ lev(i-1, j-1),\ lev(i-1, j)\} & en\ otro\ caso \end{cases}$$

```
  for (int j=1; j<=long2; j++)
    if (s1.charAt(i-1)==s2.charAt(j-1))
       lev[i][j]=lev[i-1][j-1];
    else
       lev[i][j]= 1 + Math.min(lev[i-1][j-1],
                               Math.min(lev[i-1][j],lev[i][j-1]));
  return lev[long1][long2];
}
```


168

Distancia de edición de Strings

Relación de recurrencia

Actividad 15.12. Calcula la complejidad del algoritmo en tiempo.

```
int distanciaLevenshtein (String s1, String s2){
  int long1= s1.length(); int long2= s2.length();
  int[][] lev = new int[long1+1][long2+1];
  for (int i=0; i<=long1; i++)
     lev[i][0]=i;
  for (int j=0; j<=long2; j++)
     lev[0][j]=j;
  for (int i=1; i<=long1; i++)
    for (int j=1; j<=long2; j++)
      if (s1.charAt(i-1)==s2.charAt(j-1))
         lev[i][j]=lev[i-1][j-1];
      else
         lev[i][j]= 1 + Math.min(lev[i-1][j-1],
                                 Math.min(lev[i-1][j],lev[i][j-1]));
  return lev[long1][long2];
}
```

$\Theta(N \cdot M)$

N: Longitud del s1
M: Longitud de s2

3. Distancia de Strings

1 — 2 — 3

169

Distancia de edición de Strings

Relación de recurrencia

Actividad 15.13. Calcula la complejidad del algoritmo en memoria.

```
int distanciaLevenshtein (String s1, String s2){
  int long1= s1.length(); int long2= s2.length();
  int[][] lev = new int[long1+1][long2+1];
  for (int i=0; i<=long1; i++)
     lev[i][0]=i;
  for (int j=0; j<=long2; j++)
     lev[0][j]=j;
  for (int i=1; i<=long1; i++)
    for (int j=1; j<=long2; j++)
      if (s1.charAt(i-1)==s2.charAt(j-1))
         lev[i][j]=lev[i-1][j-1];
      else
         lev[i][j]= 1 + Math.min(lev[i-1][j-1],
                                 Math.min(lev[i-1][j],lev[i][j-1]));
  return lev[long1][long2];
}
```

$\Theta(N \cdot M)$

N: Longitud del s1
M: Longitud de s2

3. Distancia de Strings

1 — 2 — 3

170

Distancia de edición de Strings

Reduciendo la complejidad en Memoria

$$lev(i,j) = 1 + \min\{\boxed{lev(i,j-1)}, \boxed{lev(i-1,j-1)}, \boxed{lev(i-1,j)}\}_{si\ s_1[i] \neq s_2[j]}$$

		0	1	2	3	4	5	6	7	8	9
			i	n	f	r	i	n	g	i	r
0		0	1	2	3	4	5	6	7	8	9
1	i	1	No necesitamos ya estos valores								8
2	n	2	1	0	1	2	3	4	5	6	7
3	f	3	2	1	0	1	2	3	4	5	6
4	l	4	3	2	1	1					
5	i	5									
6	g	6									
7	i	7									
8	r	8									

171

Distancia de edición de Strings

Relación de recurrencia

Actividad 15.14. **Optimiza el algoritmo en memoria.**

```
int distanciaLevenshtein (String s1, String s2){
  int long1= s1.length(); int long2= s2.length();
  int[][] lev = new int[2][long2+1];
  lev[0][0]=0; lev[1][0]=1;
  for (int j=0; j<=long2; j++)
     lev[0][j]=j;
  for (int i=1; i<=long1; i++)
    for (int j=1; j<=long2; j++)
      if (s1.charAt(i-1)==s2.charAt(j-1))
         lev[i % 2][j]=lev[(i-1) % 2][j-1];
      else
         lev[i % 2][j]= 1 + Math.min(lev[(i-1) % 2][j-1],
                                     Math.min(lev[(i-1) % 2][j],lev[i % 2][j-1]));
  return lev[long1 % 2][long2];
}
```

$\Theta(M)$

M: Longitud del s2

172