



Home Page

Title Page

Contents



Page 1 of 24

Go Back

Full Screen

Close

Quit

Algoritmos Voraces

Profesor: Julio César López

jlopez@eisc.univalle.edu.co

13 de noviembre de 2003



[Home Page](#)

[Title Page](#)

[Contents](#)



Page 2 of 24

[Go Back](#)

[Full Screen](#)

[Close](#)

[Quit](#)

Generalidades

- Continuamos con el objetivo de buscar buenas soluciones para problemas de optimización.
- La programación dinámica no es siempre “barata”.
- Un algoritmo voraz, toma decisiones con rapidez, encontrando óptimos locales, buscando una solución óptima global.
- Un algoritmo voraz no encuentra siempre una solución óptima, pero muchas veces la logra.

El Problema de Selección de Actividades

- ★ Suponga que se cuenta con un conjunto $S = \{a_1, \dots, a_n\}$, de **actividades** que necesitan usar un recurso que no puede ser usado sino por una actividad a la vez.
- ★ Cada actividad a_i tiene un tiempo inicial s_i y final f_i asociados, tal que $s_i \leq f_i < \infty$.
- ★ Las actividades a_i y a_j son **compatibles** si la intersección de los intervalos $[s_i, f_i) \cap [s_j, f_j) = \phi$.

El Problema de Selección de Actividades

El problema de selección de actividades:

★ **Entrada:** $S = \{a_1, \dots, a_n\}$ s_i, f_i $1 \leq i \leq n$

★ **Salida:** $A \subseteq S$: todas las actividades de A son mutuamente compatibles, y $|A|$ es el máximo posible.

Ejemplo :

Sea S un conjunto de actividades:

i	1	2	3	4	5	6	7	8	9	10	11
S_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

fig 1: Tabla de actividades

El Problema de Selección de Actividades

Las actividades representadas de forma grafica :

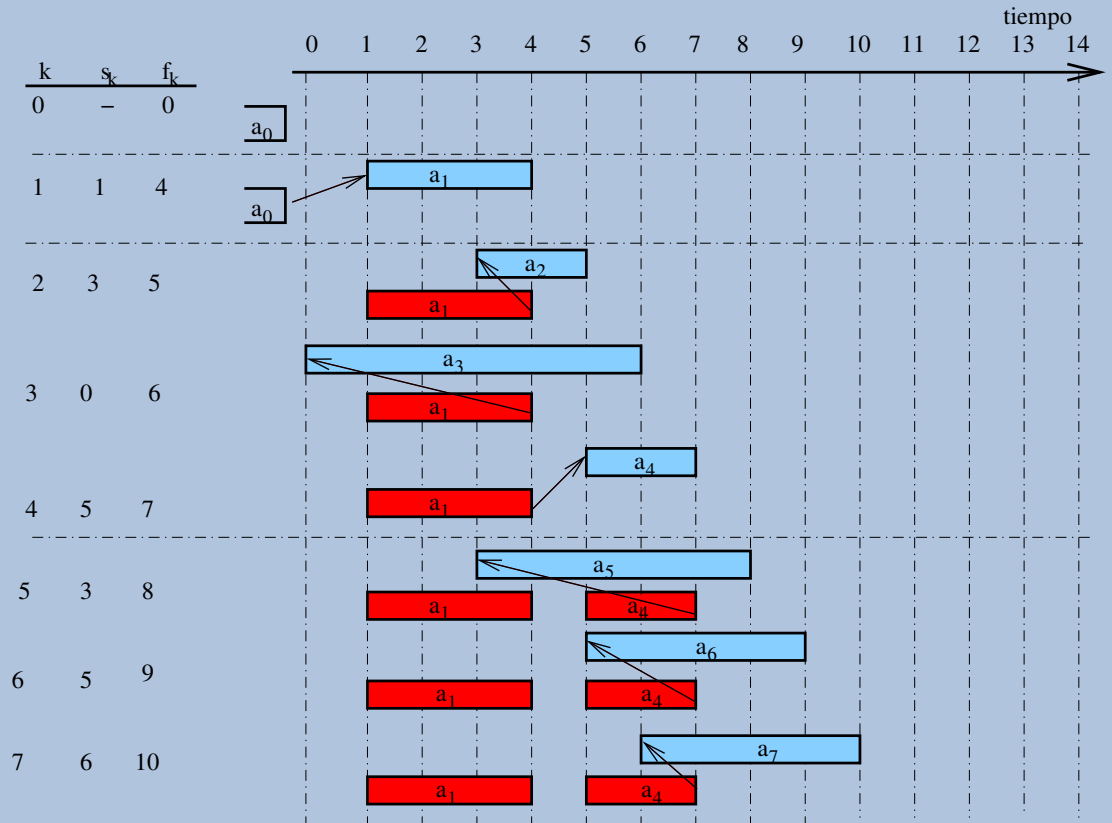


fig 1: Representación de las 11 actividades

Home Page

Title Page

Contents

◀

▶

◀

▶

Page 5 of 24

Go Back

Full Screen

Close

Quit

El Problema de Selección de Actividades



Universidad
del Valle

Home Page

Title Page

Contents



Page 6 of 24

Go Back

Full Screen

Close

Quit

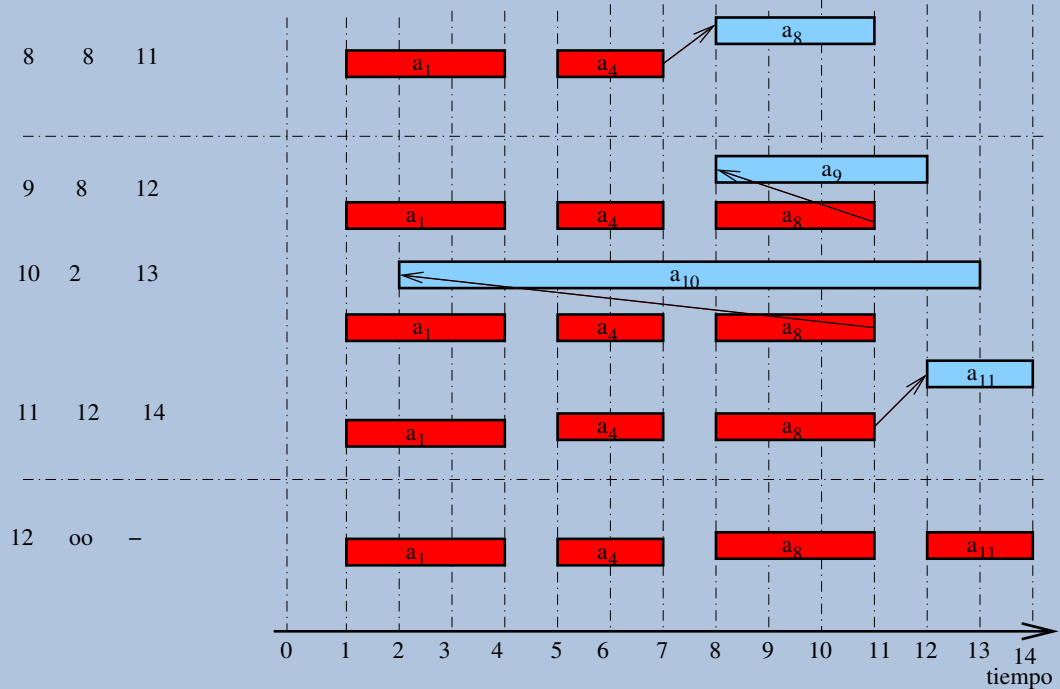


fig 1: Representación de las 11 actividades

Los subconjuntos : $\{a_3, a_9, a_{11}\}$, $\{a_1, a_4, a_8, a_{11}\}$, $\{a_2, a_4, a_9, a_{11}\}$ están compuestos de actividades mutuamente *compatibles*.



Universidad
del Valle

Home Page

Title Page

Contents



Page 7 of 24

Go Back

Full Screen

Close

Quit

El Problema de Selección de Actividades

Se ilustrará la relación entre algoritmos voraces y la programación dinámica.

Una subestructura óptima

Se define un conjunto :

$$S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$$

donde S_{ij} es el subconjunto de actividades en S que inicia después de que la actividad a_i finaliza y finaliza antes que la actividad a_j inicia.

Se asume que las actividades están ordenadas en orden creciente.

$$f_0 \leq f_1 \leq f_2 \leq \dots \leq f_n < f_{n+1}$$

El Problema de Selección de Actividades

Una solución óptima A_{ij} para S_{ij} incluyendo la actividad a_k es:

- ★ Un subconjunto de tamaño máximo A_{ij} para S_{ik}
- ★ Un subconjunto de tamaño máximo A_{kj} para S_{kj}

Cada uno con actividades mutuamente compatibles.

La idea : es formar un subconjunto de tamaño máximo A_{ij} con actividades mutuamente compatibles.

$$A_{ij} = A_{ik} \cup a_k \cup A_{kj}$$

El Problema de Selección de Actividades

Solución Recursiva

Para problema de selección de actividades tenemos que :

★ $c[i, j]$ el número de actividades en un subconjunto de tamaño máximo con actividades compatibles en S_{ij} .

★ $c[i, j] = 0$ siempre que $S_{ij} = \emptyset$.

La ecuación de recurrencia es :

$$c[i, j] = \begin{cases} 0 & \text{si } S_{ij} = \emptyset, \\ \max_{i < k < j} \{c[i, k] + c[k, j] + 1\} & \text{si } S_{ij} \neq \emptyset. \end{cases}$$

Convertir una Solución de Programación Dinámica a una Solución Voraz

Teorema:

Consideremos algún subconjunto no vacío S_{ij} y sea a_m la actividad en S_{ij} con el tiempo de finalización menor:

$$f_m = \min\{f_k : a_k \in S_{ij}\}$$

Entonces

1. La actividad a_m es utilizada por algún subconjunto de actividades tamaño máximo que son mutuamente compatibles en S_{ij} .
2. El subproblema S_{im} es vacío, de modo que la elección de a_m deja el subproblema S_{mj} como el único que no puede ser vacío.

Un Algoritmo Voraz Recursivo

El algoritmo recursivo recibe los arreglos s , f , retorna el conjunto de actividades de mayor tamaño que son compatible mutuamente en S_{ij} .

```
Sel_Act_Recursiva ( $s, f, i, j$ )  
1  $m \leftarrow i + 1$   
2 while  $m < j$  and  $s_m < f_i$   
3     do  $m \leftarrow m + 1$   
4 if  $m < j$   
5   then return  $\{a_m\} \cup \text{Sel\_Act\_Recursiva}(s, f, m, j)$   
6   else return  $\emptyset$ 
```

El procedimiento Sel_Act_Recursiva programa un conjunto de n actividades es $\theta(n)$.

Un Algoritmo Voraz Recursivo

A continuación se muestra gráficamente las operaciones del algoritmo *Sel_Act_Recursiva* :

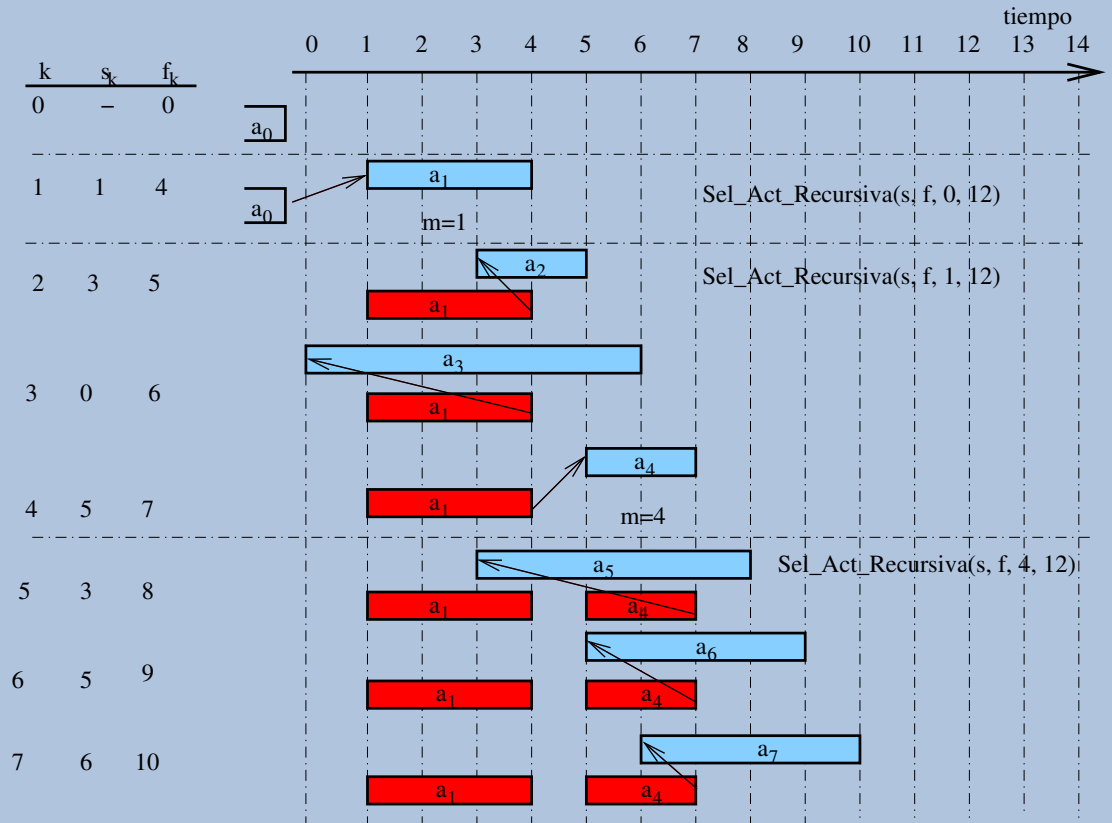


fig 2: Operación del Algoritmo Recursivo con 11 actividades

Home Page

Title Page

Contents

◀

▶

◀

▶

Page 12 of 24

Go Back

Full Screen

Close

Quit

Un Algoritmo Voraz Recursivo

Continuación operaciones realizadas

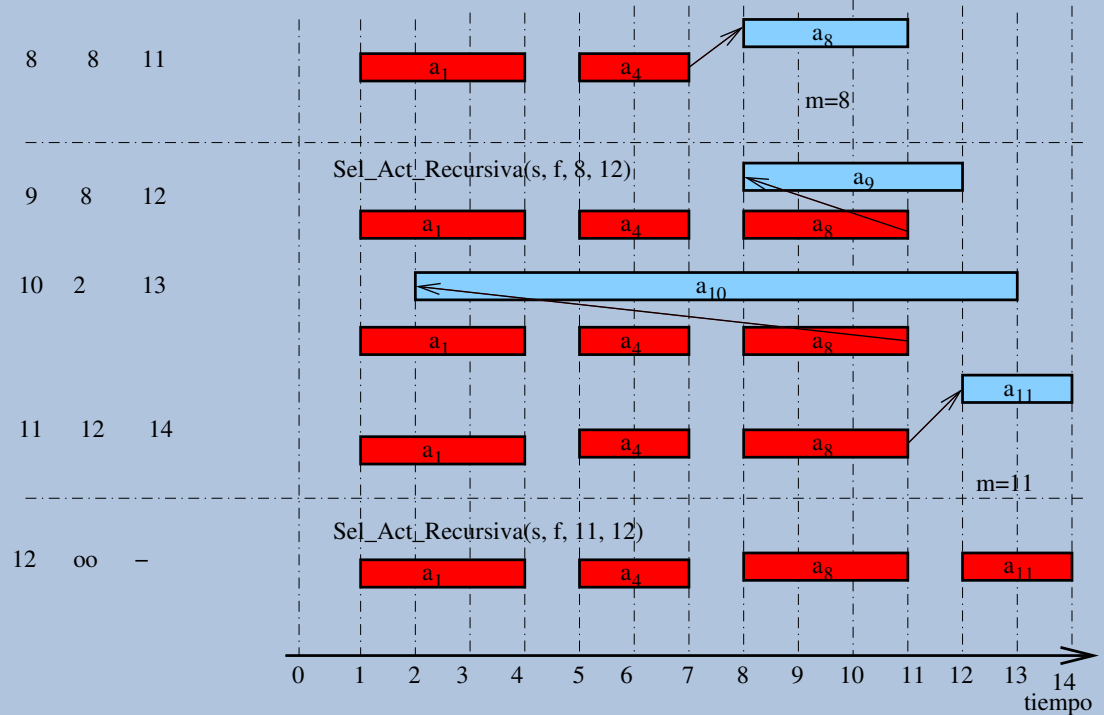


fig 2: Operación del Algoritmo Recursivo con 11 actividades



Universidad
del Valle

Home Page

Title Page

Contents



Page 14 of 24

Go Back

Full Screen

Close

Quit

Un Algoritmo Voraz Iterativo

Fácilmente se puede convertir el procedimiento recursivo anterior a uno iterativo.

Se supone que $f_1 \leq f_2 \leq \dots \leq f_n$ (si no, se ordenan primero), y que s y f son representadas por arreglos.

Seleccion_Actividades_Voraz (s, f)

```
1  $n \leftarrow \text{length}[s]$ 
2  $A \leftarrow \{a_1\}$ 
3  $i \leftarrow 1$ 
4 for  $m \leftarrow 2$  to  $n$ 
5 do if  $s_m \geq f_i$ 
6           then  $A \leftarrow A \cup \{a_m\}$ 
7                    $i \leftarrow m$ 
8 return  $A$ 
```

El procedimiento Seleccion_Actividades_Voraz planifica un conjunto de n actividades en $\theta(n)$.



Universidad
del Valle

Home Page

Title Page

Contents



Page 15 of 24

Go Back

Full Screen

Close

Quit

Características de las Estratégias Voraces

- Voraz \Rightarrow
 - ★ Serie de buenas decisiones rápidas!!
 - ★ Se escoge la mejor decisión según criterios locales.
 - ★ No siempre se encuentra el óptimo, pero puede darse.
- Cuando utilizar esta técnica?
 - ★ No hay receta, pero si el problema exhibe.
 - Propiedad de escogencia Voraz
 - Subestructura óptimaEntonces la técnica voraz puede funcionar.



[Home Page](#)

[Title Page](#)

[Contents](#)



Page 16 of 24

[Go Back](#)

[Full Screen](#)

[Close](#)

[Quit](#)

Características de las Estrategias Voraces

Propiedad de escogencia Voraz

- ★ Una solución óptima se puede hallar a partir de soluciones óptimas locales.
- ★ En programación dinámica, también se toman decisiones, pero ellas dependen de conocer las soluciones a subproblemas.
- ★ En cambio, con una técnica voraz, las decisiones se toman con la información que se tenga en el momento, y esas decisiones determinan los subproblemas que se deberán resolver.
- ★ Clave:
 - Demostrar que una solución óptima global se puede hallar a partir de decisiones óptimas locales.



Home Page

Title Page

Contents



Page 17 of 24

Go Back

Full Screen

Close

Quit

Características de las Estratégias Voraces

★ Estrategia

- Demostrar que existe al menos una solución óptima que contiene el resultado de la primera decisión “Voraz”.
- Demostrar que una vez tomada una decisión “Voraz”, lo que queda por hacer es buscar una solución óptima, con el mismo algoritmo, para un subproblema.
- Entonces, por inducción, se tendría que el algoritmo “Voraz” produce una solución óptima global.

★ **Estructura óptima:** Misma propiedad expuesta para utilizar la técnica de programación dinámica.

Características de las Estrategias Voraces

Técnicas Voraces v.s. Programación dinámica

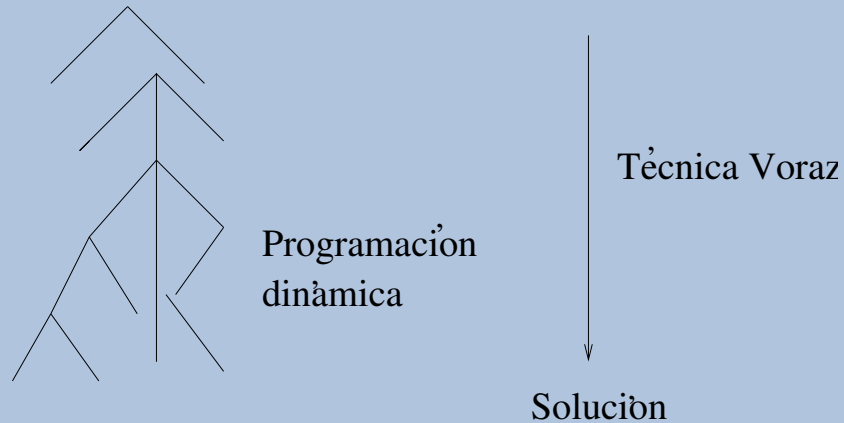


fig 3: Programación dinámica vs Voraz

Tienen en común: estructura óptima \Rightarrow puede llevar a errores



Universidad
del Valle

Home Page

Title Page

Contents



Page 19 of 24

Go Back

Full Screen

Close

Quit

Programación Dinámica vs Voraz

Consideremos los problemas:

★ Mochila 0-1

Entrada : n items y para cada item i , dos valores:

v_i : valor del item i (entero)

w_i : peso del item i (entero)

w : Máximo peso que resiste el mochila

Salida : Un subconjunto de items que, juntos, no sobrepasan el peso que resiste el morrar y de máximo valor



Home Page

Title Page

Contents



Page 20 of 24

Go Back

Full Screen

Close

Quit

Programación Dinámica vs Voraz

★ Mochila Fraccional

Entrada : Igual el anterior

Salida : Igual, pero en el mochila se pueden meter fracciones de un item

★ Ambos problemas tienen la propiedad de subestructura óptima.



Universidad
del Valle

Home Page

Title Page

Contents



Page 21 of 24

Go Back

Full Screen

Close

Quit

Programación Dinámica vs Voraz

Mochila 0-1 :

Si $A = \{i_1, \dots, i_k\}$ es una solución óptima para la entrada $I = \{1, \dots, n\}, v_i, w_i, w$

Entonces $A' = \{i_1, \dots, i_{j-1}, i_{j+1}, \dots, i_k\}$ es una solución óptima para la entrada $I' = I_{\{i_j\}}$ y $w = w' - w_j, w'_j = w_j, v'_j = v_j$

Mochila-fraccional:

Si $A = \{(p_1, i_1), \dots, (p_k, i_k)\}$ es una solución óptima para $I = \{1, \dots, n\}, w$

Entonces $A' = \{(p_1, i_1), \dots, (p_j, i_j)\}, \dots, (p_k, i_k)\}$ es una solución óptima $I' = I, w' = w - p, w'_j = w_{i_j} - p, w'_k = w_k$ si $k \neq i$ y $j v'_k = v_k$



Home Page

Title Page

Contents



Page 22 of 24

Go Back

Full Screen

Close

Quit

Programación Dinámica vs Voraz

Pero solo uno de los dos problemas tiene la propiedad de escogencia voraz: **el mochila fraccional**.

- ★ Ordenar decrecientemente los items por $\frac{v_i}{w_i}$, valor por libra de peso.
- ★ Escoger lo más que se pueda y guardarlo en la mochila, primer item, luego del segundo, etc, hasta llenar el mochila.

Programación Dinámica vs Voraz

Sin embargo, esta estrategia no funciona para **mochila 0-1**, el problema se ilustra en la siguiente figura :

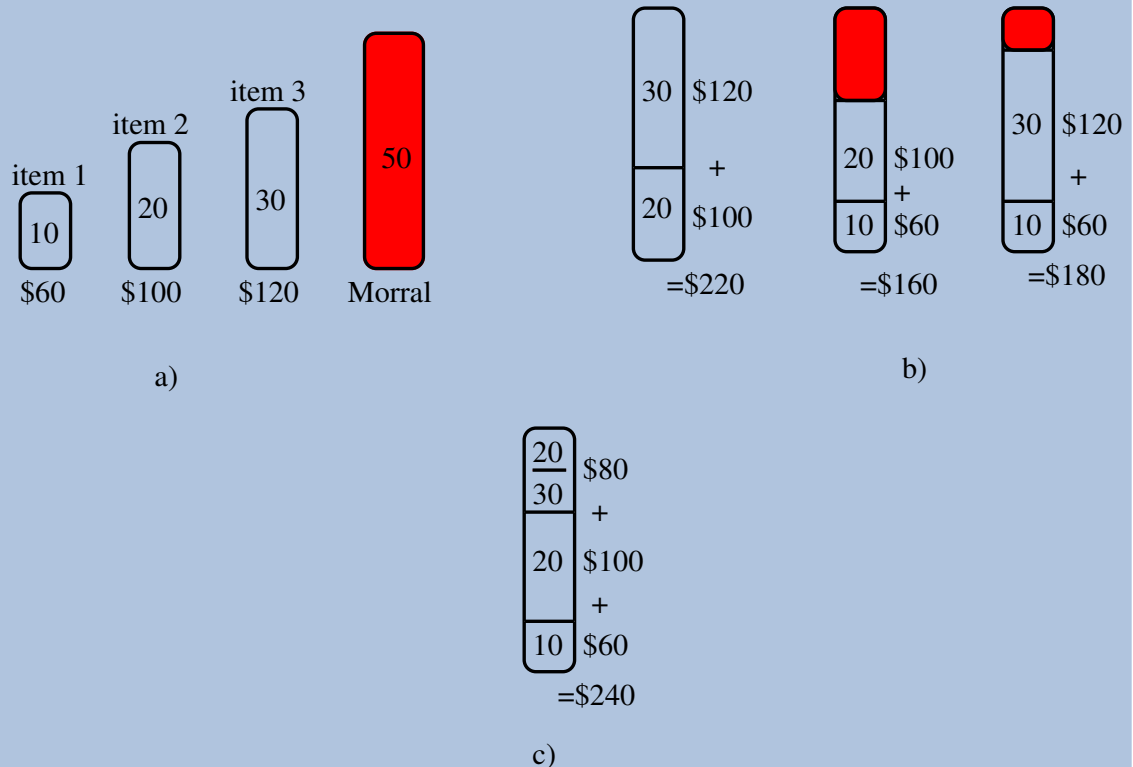


fig 4: Estrategia voraz



Home Page

Title Page

Contents



Page 24 of 24

Go Back

Full Screen

Close

Quit

Programación Dinámica vs Voraz

- ★ En el caso del mochila fraccional la solución sería: 220 que es óptima.
- ★ En el caso del **mochila 0-1** la decisión correcta no se puede tener hasta no analizar si es mejor meter un item, o dejarlo por fuera, y para este se necesita resolver subproblemas antes de tomar una decisión:
⇒ Prog. Dinámica.