Análisis y Diseño de Algoritmos II

Jesús Alexander Aranda Ph.D Robinson Duque, Ph.D
Juan Francisco Díaz, Ph. D

Universidad del Valle

jesus.aranda@correounivalle.edu.co
robinson.duque@correounivalle.edu.co
juanfco.diaz@correounivalle.edu.co
Programa de Ingeniería de Sistemas
Escuela de Ingeniería de Sistemas y Computación

Selección de actividades

Características de la programación voraz

Programación voraz

• La programación dinámica puede resultar costosa. (Pocos subproblemas repetidos)

•Otra estrategia para resolver problemas de optimización: en cada estado de la búsqueda de una solución al problema, tomar el camino (la decisión) que es el mejor en ese momento (óptima), sin tener en cuenta las soluciones a subproblemas

Programación voraz

• Un algoritmo voraz toma decisiones con rapidez sobre vistas locales → toma decisiones óptimas locales. Espera que llegue a una solución óptima global

•Un algoritmo voraz no siempre encuentra la solución óptima global

Programación voraz

Problema de selección de actividades

- Suponga que se tiene un conjunto de actividades S etiquetadas con números de $a_1 \ldots a_n$. $S=\{a_1, \ldots, a_n\}$
- Todas las actividades necesitan acceder a un mismo recurso
- Cada actividad a_i tiene asociada dos valores:

  s_i: tiempo inicial

  f_i: tiempo final

estos son los tiempos entre los cuales la actividad *debería* acceder al recurso

Programación voraz

Problema de selección de actividades

S={1,2,3}

(0,5), (1,2), (2,3) son los tiempos para las 3 actividades

Programación voraz

Problema de selección de actividades

S={1,2,3}

(0,5), (1,2), (2,3) son los tiempos para las 3 actividades

¿Cuáles son las diferentes formas de planificar las actividades?

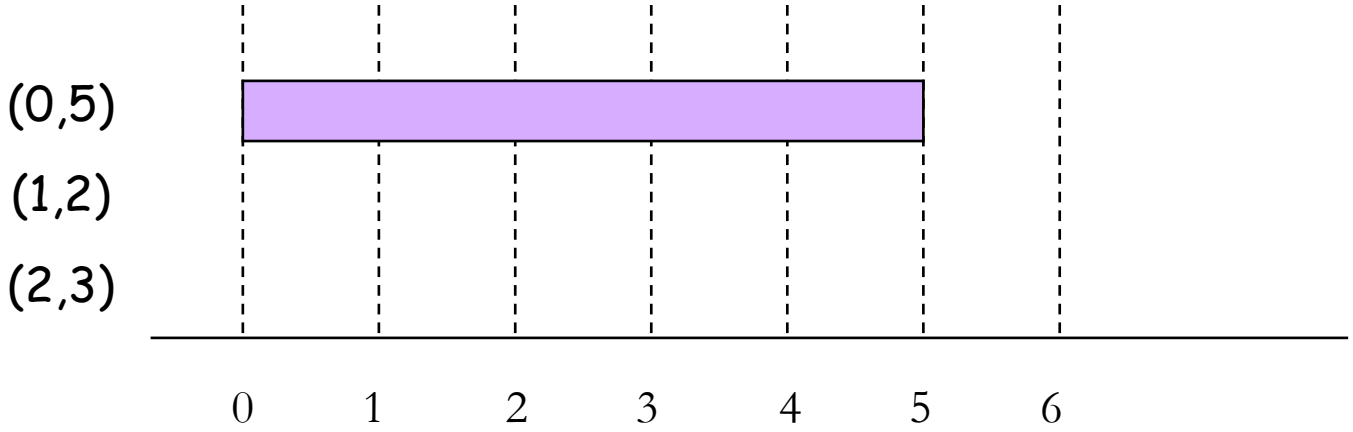
Programación voraz

Problema de selección de actividades

S={1,2,3}

(0,5), (1,2), (2,3) son los tiempos para las 3 actividades

Asignar el recurso a la actividad 1



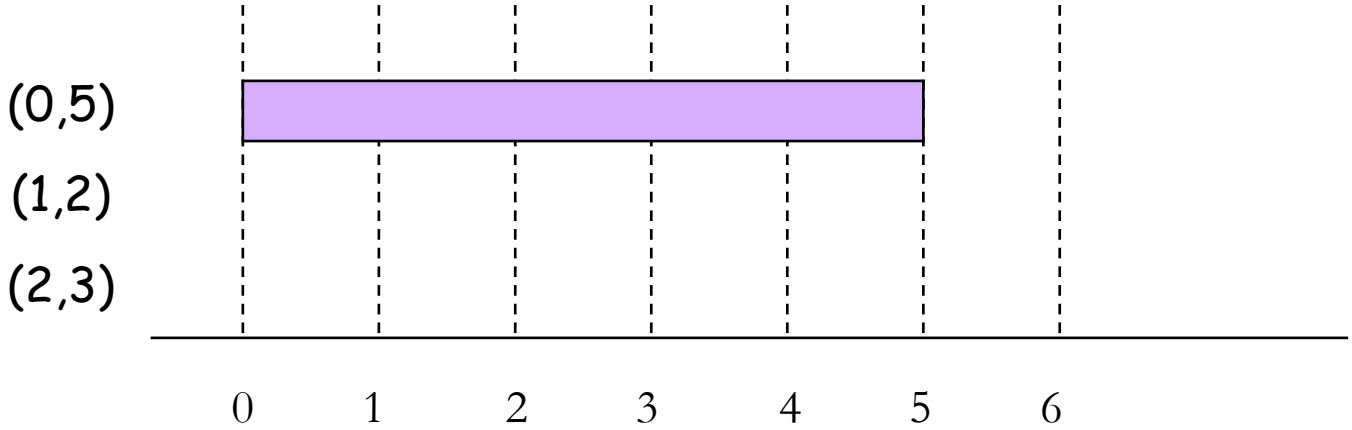
Programación voraz

Problema de selección de actividades

S={1,2,3}

(0,5), (1,2), (2,3) son los tiempos para las 3 actividades

Asignar el recurso a la actividad 1



Las actividades 2 y 3 no se podrían atender

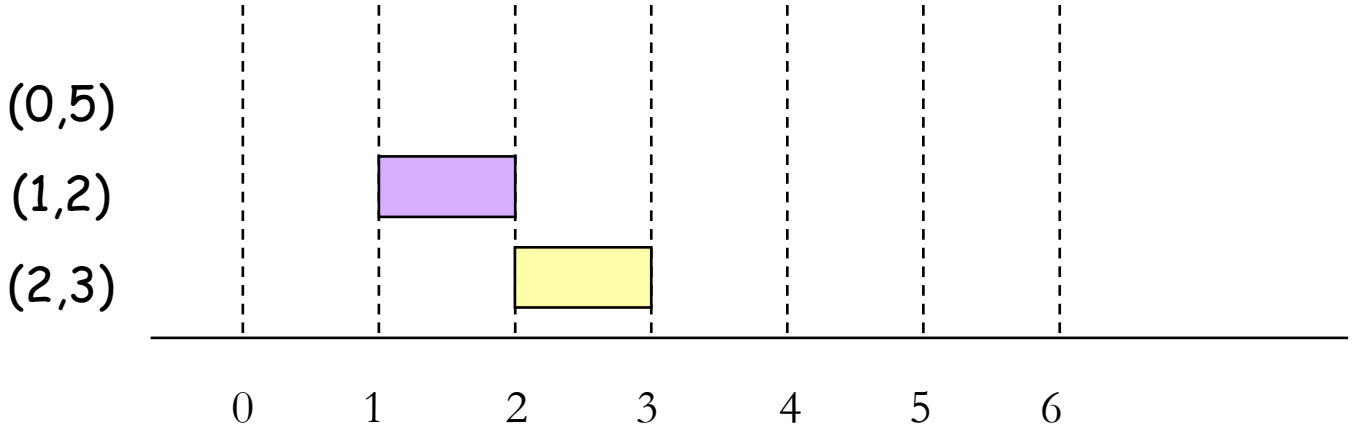
Programación voraz

Problema de selección de actividades

S={1,2,3}

(0,5), (1,2), (2,3) son los tiempos para las 3 actividades

Asignar el recurso a las actividades 2 y 3



La actividad 1 no se podría atender

Programación voraz

Problema de selección de actividades

Entrada: $S=\{a_1, \ldots, a_n\}$

Salida: $A \subseteq S$, tal que $|A|$ es máxima

Programación voraz

Problema de selección de actividades

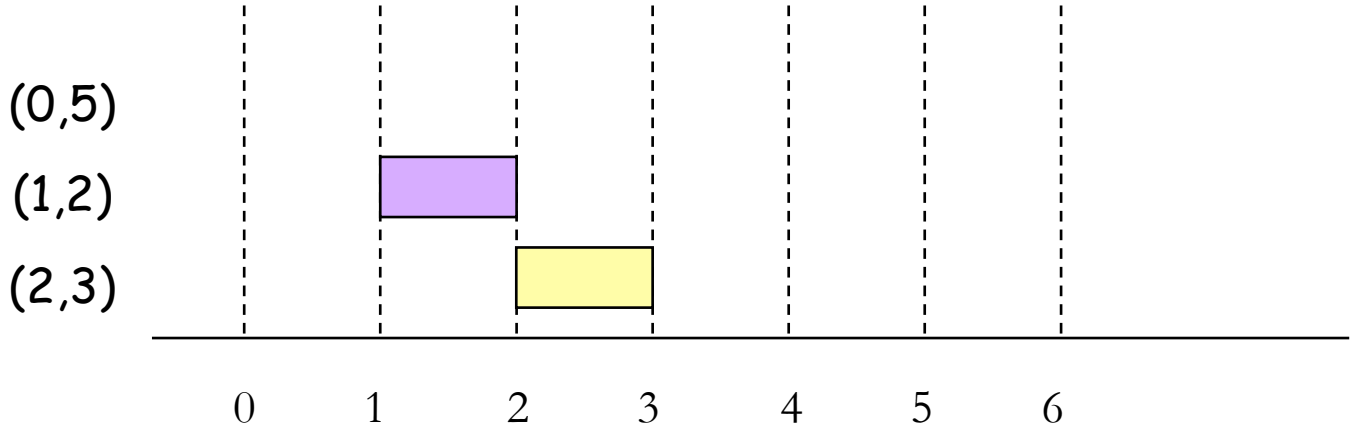
Entrada: $S=\{a_1, \ldots, a_n\}$

Salida: $A \subseteq S$, tal que $|A|$ es máxima

(maximizar la cantidad de actividades que van a usar el recurso)

Programación voraz

Problema de selección de actividades



A={2,3} es la solución óptima

Programación dinámica

Problema de Selección de Actividades (Asumiremos que las actividades están ordenadas ascendentemente según su hora de finalización)

Estrategia exhaustiva:

--- Generar todos los subconjuntos de actividades, determinar su compatibilidad y seleccionar aquella compatible con el máximo número de actividades

Costo Estrategia Exhaustiva:

----Exponencial

¿Se puede aplicar programación dinámica? Sí

Programación dinámica

Problema de Selección de Actividades (Asumiremos que las actividades están ordenadas ascendentemente según su hora de finalización)

Programación dinámica:

1. Caracterización de la estructura de una solución óptima (subestructura óptima)

Sea A_{ij} la solución al problema S_{ij} (el problema de selección de actividades considerando las actividades que inician después de que termina la actividad i y finalizan antes que inicie la actividad j .

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

Programación dinámica

Problema de Selección de Actividades (Asumiremos que las actividades están ordenadas ascendentemente según su hora de finalización)

2. Definir recursivamente el valor de una solución óptima

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset, \\ \max\{c[i,k] + c[k,j] + 1 : a_k \in S_{ij}\} & \text{if } S_{ij} \neq \emptyset. \end{cases}$$

Hay qué escoger el valor para a_k
¿Cuantas escogencias son?

¿Cuál sería la complejidad del algoritmo que calcula el costo de la solución?

Programación dinámica

Problema de Selección de Actividades (Asumiremos que las actividades están ordenadas ascendentemente según su hora de finalización)

2. Definir recursivamente el valor de una solución óptima

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset, \\ \max\{c[i,k] + c[k,j] + 1 : a_k \in S_{ij}\} & \text{if } S_{ij} \neq \emptyset. \end{cases}$$

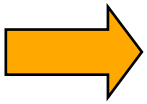
Es posible para este problema seleccionar una actividad sin tener que calcular el costo de todos los subproblemas asociados (Escogencia Voraz)

Programación voraz

Solución:

- Ordenar las actividades ascendentemente según los tiempos de finalización f_i

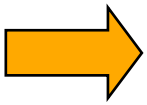
S={1,2,3}
(0,5), (1,2), (2,3)



S'={2,3,1}
(1,2), (2,3), (0,5)

- Coloque en la solución el primer recurso en la lista ordenada

S'={2,3,1}
(1,2), (2,3), (0,5)



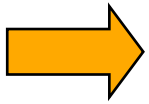
A={2}

Programación voraz

Solución:

- Coloque en la solución A, el recurso en S' que tiene tiempo de inicio menor o igual que el tiempo final del recurso que se acaba de planificar

S'={2,3,1}
(1,2), (2,3), (0,5)

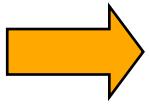


Programación voraz

Solución:

- Coloque en la solución A, el recurso en S' que tiene tiempo de inicio menor o igual que el tiempo final del recurso que se acaba de planificar

S'={2,3,1}
(1,2), (2,3), (0,5)



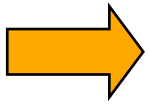
A={2,3}

Programación voraz

Solución:

•Coloque en la solución A, el recurso en S' que tiene tiempo de inicio menor o igual que el tiempo final del recurso que se acaba de planificar

S'={2,3,1}
(1,2), (2,3), (0,5)



A={2,3}

¿Por qué es una estrategia voraz?

- Se toma una decisión óptima local en cada estado de la solución
- La decisión no depende de solucionar primero subproblemas relacionados

Programación dinámica

Algoritmo Voraz

```
RECURSIVE-ACTIVITY-SELECTOR (s, f, k, n)
1 m = k + 1
2 while m ≤ n and s[m] < f [k]  // find the first activity in S_k to finish
3     m = m + 1
4 if m ≤ n
5     return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR (s, f, m,
          n)
6 else return ∅
```

Llamado inicial RECURSIVE-ACTIVITY-SELECTOR(s,f,0,n)

Programación voraz

¿Cuándo utilizar una estrategia voraz?

Cuando el problema exhiba:

- *Propiedad de escogencia voraz*
- *Subestructura óptima*

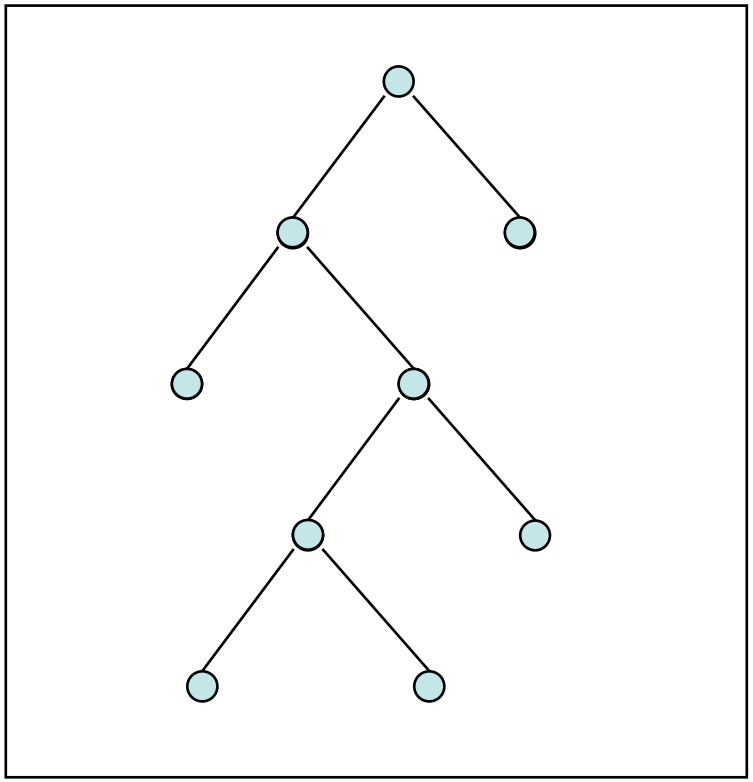
Programación voraz

¿Cuándo utilizar una estrategia voraz?

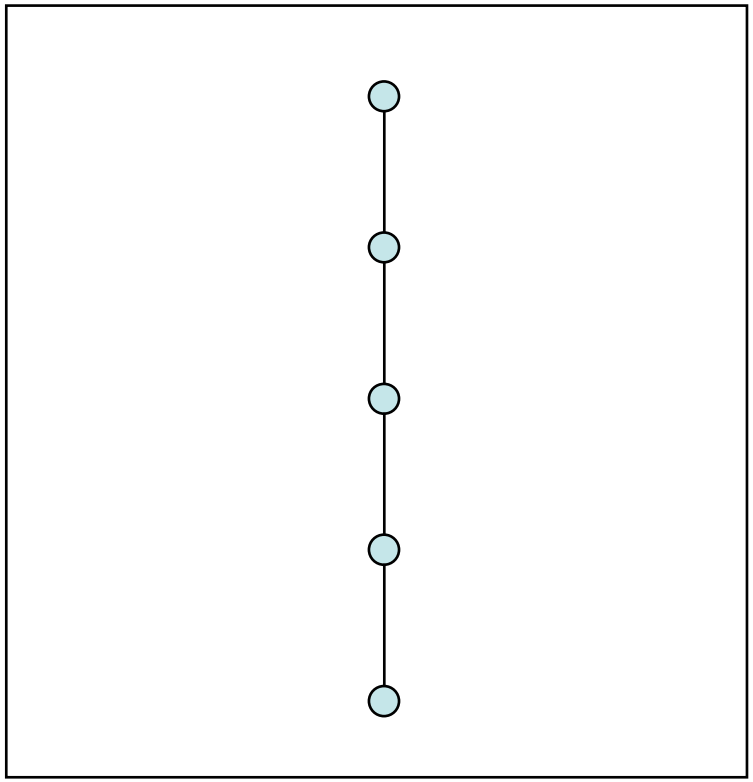
Cuando el problema exhiba:

- *Propiedad de escogencia voraz:* una solución óptima se puede hallar a partir de soluciones óptimas locales
- *Subestructura óptima:* igual que en programación dinámica

Programación voraz



Programación dinámica



Programación voraz

Programación voraz

Problema: Mochila 0-1

Se tienen N objetos y una mochila de capacidad (de peso) M, cada objeto tiene un peso w_i, $1 \leq i \leq N$. Cada objeto puede estar, o no, en la mochila. Ademas, se tiene un beneficio b_i por cada objeto

El problema consiste en maximizar el beneficio. La solución se representa indicando para cada objeto si se debe colocar o no en la mochila

Programación voraz

De manera formal, el problema consiste en encontrar $\langle x_1, x_2, \ldots, x_n \rangle$ tal que:

$$\sum_{1 \le i \le N} b_i x_i$$ sea máximo, sujeto a

$$\sum_{1 \le i \le N} w_i x_i \le M$$

$x_i \in \{0,1\}$, donde 0 significa que el objeto i no se coloca en la mochila y 1 que si

Programación voraz

N=3, M=9, b=<10,6,8>, w=<3,4,5>

<1,0,1> es una solución que indica colocar en la mochila los objetos 1 y 3, esto implica un beneficio de 18

<1,1,0> es una solución que indica colocar en la mochila los objetos 1 y 2, esto implica un beneficio de 16

<0,1,1> es una solución que indica colocar en la mochila los objetos 2 y 3, esto implica un beneficio de 14

Programación voraz

Estrategia voraz: seleccionar el ítem que tiene mayor beneficio por peso, esto es, b_i/w_i sea mayor

Programación voraz

Estrategia voraz: seleccionar el ítem que tiene mayor beneficio por peso, esto es, b_i/w_i sea mayor

N=3, M=9, b=<10,6,8>, w=<3,4,5>

Beneficio/peso=<10/3, 6/4, 8/5> = <3.3 , 1.5, 1.6 >

Seleccionar el item1, luego el item3 y por último el item2 (si caben)

Programación voraz

Estrategia voraz: seleccionar el ítem que tiene mayor beneficio por peso, esto es, b_i/w_i sea mayor

N=3, M=9, b=<10,6,8>, w=<3,4,5>

Beneficio/peso=<10/3, 6/4, 8/5> = <3.3 , 1.5, 1.6 >

Seleccionar el item1, luego el item3 y por último el item2 (si caben)

Solución: <1,0,1>

Beneficio=10+8

Programación voraz

Estrategia voraz: seleccionar el ítem que tiene mayor beneficio por peso, esto es, b_i/w_i sea mayor

N=3, M=50, b=<60,100,120>, w=<10,20,30>

Beneficio/peso=<60/10, 100/20, 120/30> = <6, 5, 4 >

Seleccionar el item1, luego el 2

Solución: <1,1,0>

Beneficio=60+100=160

Programación voraz

Estrategia voraz: seleccionar el ítem que tiene mayor beneficio por peso, esto es, b_i/w_i sea mayor

N=3, M=50, b=<60,100,120>, w=<10,20,30>

Beneficio/peso=<60/10, 100/20, 120/30> = <6, 5, 4 >

La solución óptima es: <0,1,1>

Beneficio=100+120=220