Algoritmos y Estructuras de Datos II
Clase 10

Carreras:

Licenciatura en Informática

Ingeniería en Informática

2024

Unidad III

Técnicas de diseño de algoritmos

Algoritmos greedy (2)

Problema de la Mochila

Este problema es una variante del problema de carga ya estudiado.

- Se tiene ***n*** ***objetos*** y una mochila para llevarlos.
- Cada objeto tiene un ***peso*** y un ***beneficio*** asociado
- La mochila puede cargar un ***peso máximo*** dado.
- Nuestro objetivo es llenar la mochila de tal manera que se ***maximice el beneficio de los objetos transportados***, respetando la limitación de la capacidad impuesta.

Problema de la Mochila 0/1

Datos: n objetos con sus pesos y beneficios y capacidad de carga:

M= *capacidad* máxima de la mochila

Para $i=1,2,3,\dots n$

p_i= *peso* de cada objeto i

b_i= *beneficio* asociado al objeto i

Solución: vector X , $i=1,2,3,\dots n$

$x_i = 0$ si el objeto i no va en la mochila

$x_i = 1$ si el objeto i se carga en la mochila

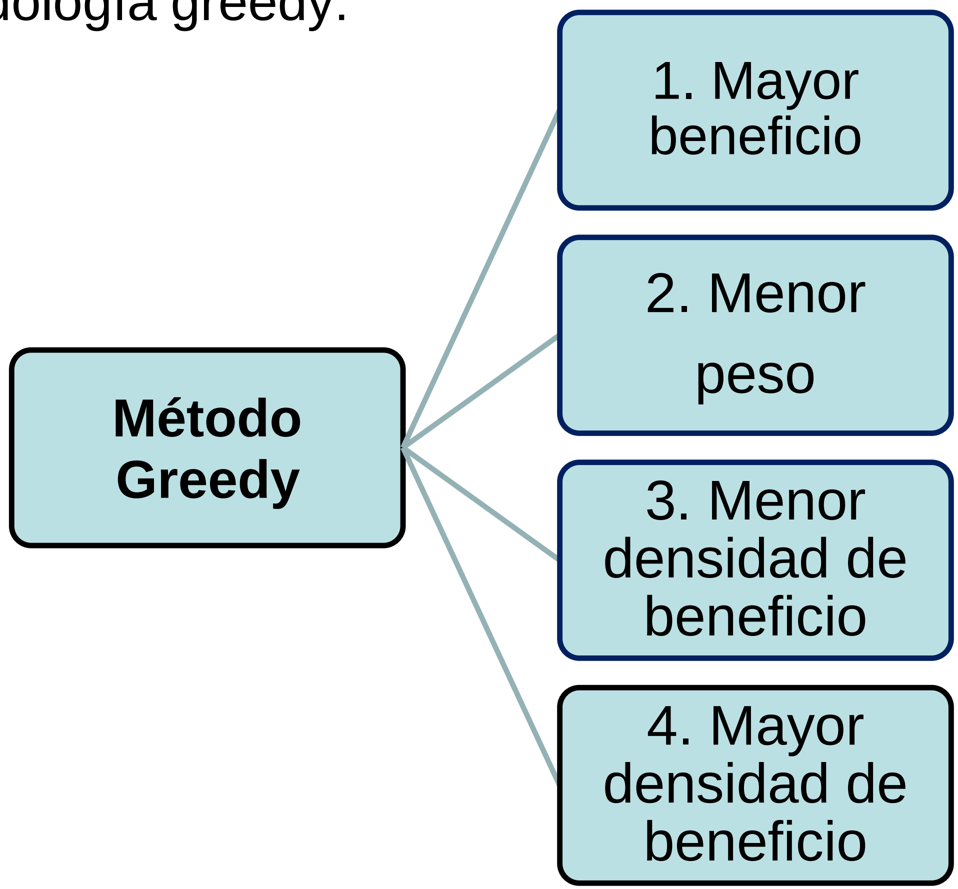
Objetivo: Maximizar el beneficio total de la carga: $\sum_{i=1}^{n} b_i x_i$

Restricción: La capacidad de la mochila no debe ser superada:

$$\sum_{i=1}^{n} p_i x_i \leq M$$

Problema de la Mochila 0/1

Hay varias estrategias posibles para resolver este problema con la metodología greedy:



Problema de la Mochila 0/1

Aplicación de las estrategias posibles para resolver este problema .

1. Cargar la mochila aplicando ***método Greedy al beneficio*.**

- Cargar la mochila en **orden decreciente** de beneficios.

- Esta estrategia no garantiza solución óptima.

Por ejemplo: n=3, M=105 , p=[100,10,10],
b=[20,15,15]

Solución Greedy en beneficio: x=[1,0,0] , Beneficio Total = 20
Solución óptima : x=[0,1,1] , Beneficio Total = 30

Problema de la Mochila 0/1

2. Otra alternativa es el ***método Greedy para el peso*.**

- Cargar la mochila en **orden creciente** de pesos.

- Esto da una solución optima en el ejemplo anterior pero no garantiza solución óptima siempre.

Por ejemplo: n=2, M=25, p=[10,20],
b=[5,100]

Solución Greedy en peso : x=[1,0] , Beneficio Total = 5
Solución óptima : x=[0,1] , Beneficio Total = 100.

Problema de la Mochila 0/1

3. Aplicar ***método Greedy a la densidad de beneficio*.**

- Cargar la mochila considerando los objetos en **orden creciente** de densidad de beneficio (cociente bi/pi) .

- Esta estrategia tampoco garantiza la solución optima.

Por ejemplo: n=3, M=30, p=[20,15,15],
b=[40,45,45],
b/p=[2, 3, 3]

Solución Greedy en densidad: x=[1,0,0] , Beneficio Total = 40
Solución optima : x=[0,1,1] , Beneficio Total = 90

Problema de la Mochila 0/1

4. Aplicar ***método Greedy a la densidad de beneficio*** en otro orden.

- Cargar la mochila considerando los objetos en **orden decreciente** de densidad de ganancias (cociente bi/pi) .

- Esta estrategia tampoco garantiza la solución optima.

Por ejemplo: n=5, M=100, p=[30,10,20,50,10],
b=[66,20,30,60,10],
b/p=[2.2, 2.0, 1.5, 1.2, 1.0]

Solución Greedy en densidad: x=[1,1,1,0,1] , Beneficio Total = 126
Solución óptima : x=[1,1,0,1,1] , Beneficio Total = 156

Problema de la Mochila 0/1

Se puede demostrar que la técnica greedy aplicada en el problema de la mochila **NO funciona**:

greedy en beneficio,

greedy en peso,

greedy de beneficio/peso,

greedy peso/beneficio,

Etc,etc,etc…

Solamente si se supone que *los* ***n objetos se pueden partir*** de manera que ***se pueda llevar una fracción de cada objeto***, entonces si se puede implementar una estrategia greedy que funcione.

Problema de la Mochila fracción

Datos:

n objetos para cargar en la mochila
M= *capacidad* máxima de la mochila
Para cada objeto $i=1,2,3,\ldots n$
pi= *peso* de cada objeto i
bi= *beneficio* asociado al objeto i

Solución: vector X , $i=1,2,3,\ldots n$
$0 \leq x_i \leq 1$ representa *la parte del objeto i* que va en la mochila

Objetivo: Maximizar el beneficio total de la carga: $\sum_{i=1}^{n} b_i x_i$

Restricción: La capacidad de la mochila no debe ser superada:

$$\sum_{i=1}^{n} p_i x_i \leq M$$

Funcion **Mochila** (p,b,n,M): vector x vector x entero≥0 x entero≥0 → vector

Para i=1,n hacer

    X(i)←0

peso←0

Mientras peso < M hacer

    k← índice i *//del objeto de mayor bi/pi entre aquellos objetos tal que X(i)=0*

    si peso+p(k)≤M entonces *// lleva completo*

        X(k)←1

        peso←peso+p(k)

    sino *// lleva una porción*

        X(k)←(M-peso)/p(k)

        peso←M

Fin mientras

Retorna X

Fin

Este algoritmo greedy garantiza que X es una solucion optima al problema de la mochila con fraccion

Problema de la Mochila fracción

Ejemplo:
Objetos n=5
Capacidad M=100

Peso	30	10	20	50	40
Beneficio	66	20	30	60	40
Beneficio/Peso	2.2	2.0	1.5	1.2	1.0

Solución X=	x1	x2	x3	x4	x5	Beneficio Total	
Greedy en:							
menor peso	1	1	1	0	1	156	
mayor beneficio	1	0	0	1	0.5	146	
mayor beneficio/peso	1	1	1	0.8	0	164	óptimo

La técnica greedy en beneficio/peso seleccionará primero el objeto 1, luego el 2 y el 3 y finalmente llenará la mochila con 4/5 del objeto 4, completando así su capacidad. La solución obtenida tiene un beneficio de 164, óptimo para este problema.

Problema de la Mochila fracción

El siguiente teorema muestra que la técnica greedy con la selección del objeto que maximice el beneficio por unidad de peso lleva a la solución óptima.

Teorema: si se ordenan los objetos en orden decreciente de beneficio por unidad de peso b_i/p_i y se cargan en la mochila enteros mientras se pueda y cuando no quede capacidad se carga la fracción correspondiente, la solución es óptima.

Demostración:
Se puede suponer que los objetos disponibles están ordenados por valor decreciente del cociente:

$$\frac{b_1}{p_1} \geq \frac{b_2}{p_2} \geq \cdots \frac{b_n}{p_n}$$

Problema de la Mochila fracción

Sea $X = (x_1, x_2, \ldots, x_n)$ la solución encontrada por la técnica greedy. Se puede suponer entonces que no todos los x_i son *1.*

$$X=(1,1,1,1,\ldots,\ \mathbf{x_j}\ ,0,0,0,0)$$

Sea j el menor índice tal que $\mathbf{x_j \neq 1}$, entonces del algoritmo se concluye que:

- Si $i<j$ entonces $x_i=1$
- Si $i=j$ entonces $0<x_i<1$
- Si $i>j$ entonces $x_i=0$

Por construcción del algoritmo el peso de la solución: $\sum_{i=1}^{n} x_i p_i = M$

El beneficio de la solución X será: $B(X) = \sum_{i=1}^{n} x_i b_i$

Problema de la Mochila fracción

Sea otra solución factible $\boldsymbol{Y}$ encontrada por cualquier técnica:

$$Y = \left(y_1, y_2, \ldots, y_n\right)$$

Por ser solución se cumple que: $\sum_{i=1}^{n} y_i p_i \leq M$

Restando los pesos: $\underbrace{\sum_{i=1}^{n} x_i p_i}_{=M} - \underbrace{\sum_{i=1}^{n} y_i p_i}_{\leq M} = \sum_{i=1}^{n} (x_i - y_i) p_i \geq 0$ **(*)**

El beneficio de la solución Y será: $B\left(Y\right) = \sum_{i=1}^{n} y_i b_i$

Problema de la Mochila fracción

La diferencia de los beneficios de la dos soluciones:

$$B(X)-B(Y)=\sum_{i=1}^{n}(x_i-y_i)b_i=\sum_{i=1}^{n}\overbrace{(x_i-y_i)\frac{b_i}{p_i}}\,p_i$$

Para analizar este producto: $(x_i-y_i)\dfrac{b_i}{p_i}$ se consideran 3 casos:

- ***i<j*** → $x_i=1$ → $(x_i-y_i)\geq 0$ y además $\dfrac{b_i}{p_i}\geq\dfrac{b_j}{p_j}$ por la ordenación elegida
- ***i>j*** → $x_i=0$ → $(x_i-y_i)\leq 0$ y además $\dfrac{b_i}{p_i}\leq\dfrac{b_j}{p_j}$ por la ordenación elegida
- ***i =j*** → $\dfrac{b_i}{p_i}\equiv\dfrac{b_j}{p_j}$

Problema de la Mochila fracción

En los 3 casos se tiene que: $$(x_i - y_i)\frac{b_i}{p_i} \geq (x_i - y_i)\frac{b_j}{p_j}$$

de (*) :
$$\sum_{i=1}^{n}(x_i - y_i)p_i \geq 0$$

Aplicando a la diferencia de beneficios:

$$B(X) - B(Y) = \sum_{i=1}^{n}\overbrace{(x_i - y_i)\frac{b_i}{p_i}}\, p_i \geq \sum_{i=1}^{n}\overbrace{(x_i - y_i)\frac{b_j}{p_j}}\, p_i = \frac{b_j}{p_j}\sum_{i=1}^{n}(x_i - y_i)p_i \geq 0$$

Entonces: ***B(X) ≥ B(Y)***

Con lo cual queda demostrado que ninguna otra solución puede tener un valor mayor de ***B(X)*** por lo que **la solución *X* es óptima.**

Problema de la Mochila fracción

Costo del algoritmo

Si los objetos ya están ordenados por orden decreciente de b_i/p_i entonces el algoritmo greedy requiere un tiempo **O(n)** para armar el vector solución.

Si se le agrega el proceso de ordenación entonces el tiempo total es:

$$T(n) \in O(n \log n)$$

El tiempo se puede disminuir si en lugar de ordenar los b_i/p_i se arma una PQ de mayor con ellos y se la implementa con un montículo.

Planificación

Los problemas de planificación, o atención de tarea, (*job scheduling*) tienen distintos objetivos como buen rendimiento del sistema o tiempo de espera mínimo. Entre los esquemas mas usados están:

- Planificación FIFO, *First In First Out*
- Planificación con Plazo Fijo
- Planificación por Turno Rotatorio, *Round Robin*
- Planificación por Prioridad al más corto, *Short Job First*
- Planificación por Prioridad al Tiempo Restante más Corto, *Short Remaining Time First*
- Planificación a la Tasa de Respuesta más Alta, *Highest Response Ratio Next*
- Planificación por el Comportamiento.

Planificación

Se plantean problemas de optimización de mínimo o de máximo que se pueden resolver con la técnica Greedy:

Algunos algoritmos Greedy tratan de planificar la ejecución de tareas con uno o más procesadores o máquinas o servidores de modo que se *minimice el tiempo medio que usa cada tarea en el sistema.*

En otros casos se puede resolver por la técnica Greedy si cada tarea tiene un beneficio asociado y el objetivo es *maximizar la rentabilidad* concluyendo las tareas en un plazo establecido.

Planificación - Un servidor

Problema: Minimización del tiempo en el sistema.

Considere que hay disponible ***un solo servidor*** que tiene que atender n clientes.

- El tiempo de atención que necesita cada cliente se conoce de antemano.
- Se quiere *minimizar el tiempo medio invertido por cada cliente* en el sistema.

Datos:

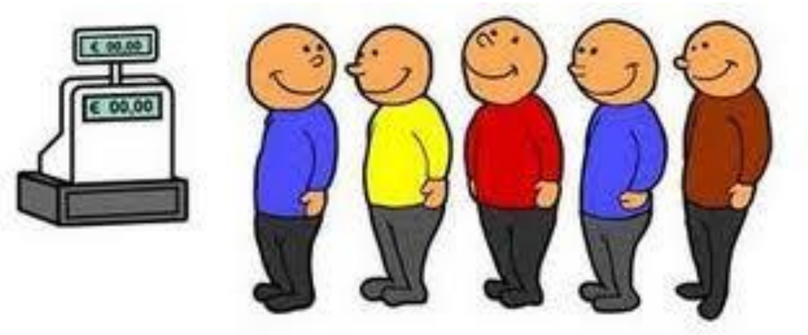
n : número de clientes, numerados con $i=1,n$

t_i : tiempo de atención del cliente i

E_i: tiempo de espera del cliente i

Objetivo:

Minimizar: $$E = \sum_{i=1}^{n} E_i$$



Planificación - Un servidor

Ejemplo: n=3 clientes: C1, C2, C3, tiempos: t1=5, t2=10, t3=3
Tiempo total de atención para estos clientes = $\sum$ ti = 18 .
Existen n!=6 posibles esquemas de orden de atención.

ORDEN	**Tiempo de Espera**			
	C1	**C2**	**C3**	**TIEMPO MEDIO**
C1C2C3	5	5+10	5+10+3	38 / 3
C1C3C2	5	5+3+10	5+3	31 / 3
C2C1C3	10+5	10	10+5+3	43 / 3
C2C3C1	10+3+5	10	10+3	41 / 3
C3C1C2	3+5	3+5+10	3	29 / 3
C3C2C1	3+10+5	3+10	3	34 / 3

Optimo

El algoritmo greedy atiende a los clientes en orden creciente de *ti* y garantiza siempre la solución óptima en este problema.

Planificación - Un servidor

El algoritmo greedy atenderá a los clientes en orden creciente de ti.

El que requiera menos tiempo se atenderá primero, luego el siguiente y así.

En cada paso se agrega al final de la planificación al cliente que requiera el menor tiempo de servicio entre los restantes.

El tiempo total será el mismo pero el tiempo medio de espera de los clientes será el menor.

Sean los n clientes y sus respectivos tiempos: $t_1, t_2, \ldots t_n$, y sea E_i el tiempo que espera el cliente i-ésimo.

Lo que se consigue con este algoritmo es *minimizar la expresión*:

$$E(n) = \sum_{i=1}^{n} E_i$$

Planificación - Un servidor

ALGORITMO GREEDY: la permutación óptima es la que organiza a los clientes por orden creciente de los tiempos.

Teorema: el algoritmo greedy para planificación es óptimo.
Demostración:
Si se considera que los clientes se atienden en el orden *(1,2,...n)*, el tiempo de espera de cada cliente:

$$E_1 = t_1$$

$$E_2 = t_1 + t_2$$

$$\dots$$

$$E_n = t_1 + t_2 + \dots + t_n$$

Se quiere probar que la permutación optima es aquella en que los clientes se atienden en orden creciente de sus tiempos.

Planificación - Un servidor

El tiempo de espera de los n clientes de la solución greedy:

$$E(n) = \sum_{i=1}^{n} E_i = t_1 + (t_1 + t_2) + (t_1 + t_2 + t_3) + \ldots$$

$$E(n) = nt_1 + (n-1)t_2 + (n-2)t_3 + \ldots$$

$$E(n) = \sum_{i=1}^{n} (n-i+1)t_i$$

Planificación - Un servidor

Sea $X = (x1,x2,...,xn)$ a una permutación de los elementos $(1,2,...,n)$, y sean $(s1,s2,...,sn)$ sus respectivos tiempos de ejecución, $(s1,s2,...,sn)$ va a ser una permutación de los tiempos originales $(t1,t2,...,tn)$.

El tiempo de espera de X : $$E(X) = \sum_{i=1}^{n} (n - xi + 1) si$$

Se puede suponer que ***no está ordenada*** en orden creciente de tiempo, es decir, que existen dos números $xi < xj$ tales que $si > sj$.

$$xj - xi > 0 \quad \text{y} \quad si - sj > 0 \qquad (*)$$

Sea $Y = (y1,y2,...,yn)$ la permutación obtenida a partir de X intercambiando solamente los dos elementos xi con xj :

$$yi = xj, \quad yj = xi. \qquad (**)$$

en los otros casos (si $k \neq i$ y $k \neq j$):

$$yk = xk$$

Planificación - Un servidor

El tiempo de espera de la solución X es: $E(X) = \sum_{i=1}^{n} (n - xi + 1) si$

El tiempo de espera de la solución Y será:

$$E(Y) = (n - yj + 1)sj \quad + \quad (n - yi + 1)si + \sum_{k=1, k \neq i, k \neq j}^{n} (n - k + 1)sk$$

(**) $yj = xi$

(**) $yi = xj$

$$E(Y) = (n - xi + 1)sj \quad + \quad (n - xj + 1)si + \sum_{k=1, k \neq i, k \neq j}^{n} (n - k + 1)sk$$

de (*)
$xj - xi > 0$
$si - sj > 0$

La diferencia del tiempo de espera de las soluciones:

$$E(X) - E(Y) = (n - xi + 1)(si - sj) + (n - xj + 1)(sj - si) = (xj - xi)(si - sj) > 0$$

De modo que resulta: $E(X) - E(Y)) > 0$

Se ha demostrado que: $\mathbf{E(Y)<E(X)}$**.** Esto indica que mientras más ordenada (según el criterio dado) esté la permutación como en el caso de la solución Y, menor tiempo de espera supone.

Planificación - Un servidor

En consecuencia, el algoritmo greedy consiste en atender a los clientes en orden inverso a su tiempo requerido de atención.

Todos los esquemas de atención que atiendan por orden no decreciente de tiempo de servicio obtienen soluciones óptimas.

Con esto conseguirá minimizar el tiempo medio de espera de los clientes, tal como se ha probado.

Costo del algoritmo:

El proceso de ordenar los tiempos en orden no decreciente es O(n.logn). Luego se repite el proceso de elección n veces.

De modo que el tiempo total del algoritmo es:

$$\mathbf{T(n) \in O(n.logn)}$$

Planificación - Varios servidores

Si se aumenta el servidores, con lo que ahora se dispone de un total de S servidores para realizar las n tareas.

En este caso también se tiene que ***minimizar el tiempo medio de espera de los clientes***, pero con la diferencia que ahora existen S servidores dando servicio simultáneamente.

Planificación - Varios servidores

Basándose en el método utilizado anteriormente se plantea la estrategia de atención.

La forma óptima de atender los clientes en este caso es la siguiente:

- En primer lugar, se ordenan los clientes por ***orden creciente*** de tiempo de servicio.

- Una vez ordenados, se van asignando los clientes por orden, siempre al servidor menos ocupado.

- En caso de haber varios con el mismo grado de ocupación, se elige el de número menor.

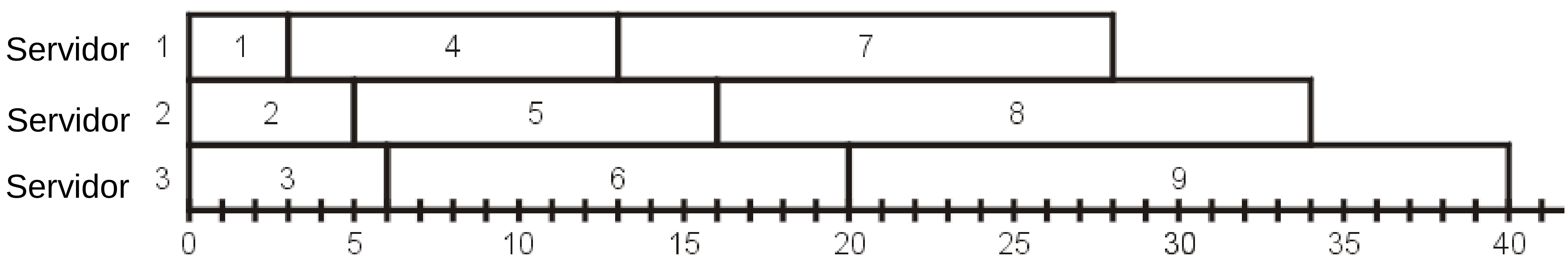
Planificación - Varios servidores

Si los clientes están ordenados de forma que:

$$ti \leq tj \quad \text{si} \quad i < j,$$

Un algoritmo greedy asigna al servidor k las tareas k, $k+S$, $k+2S$, ...

Por ejemplo:

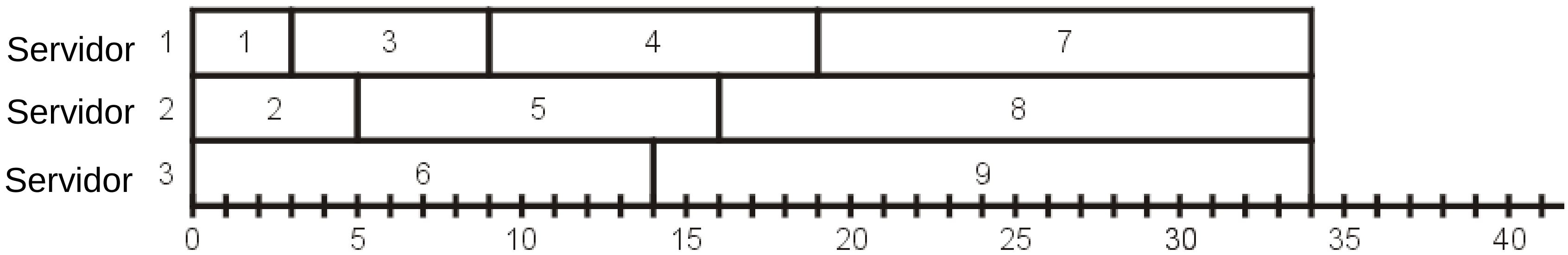


Esta distribución *minimiza* el tiempo medio de espera de los clientes.

Planificación - Varios servidores

Otro problema distinto se presenta cuando se quiere *minimizar el tiempo de ocupación de los servidores.*

Para el ejemplo anterior resulta en una distribución:



Se podrá aplicar una estrategia greedy?