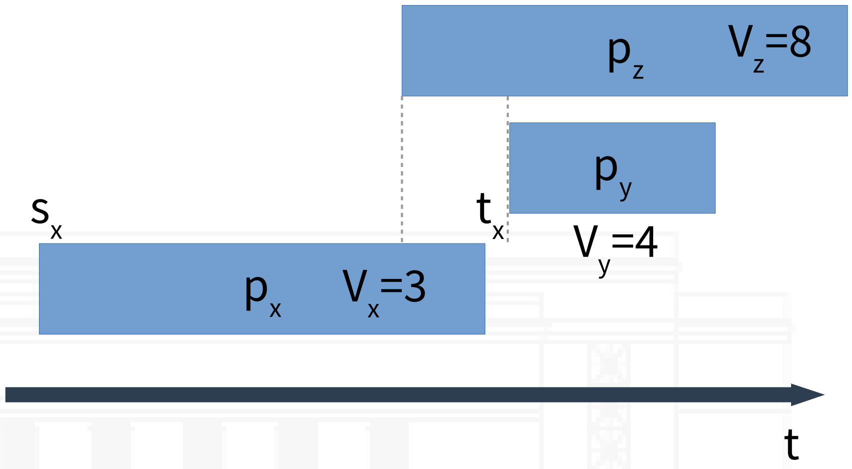
Programación dinámica: Weighted Interval Scheduling

Teoría de Algoritmos I (75.29 / 95.06)

Ing. Víctor Daniel Podberezski
vpodberezski@fi.uba.ar

Weighted Interval Scheduling



Si bien puedo seleccionar x e y con un valor de 7, es preferible z con un valor total de 8

Sea

P un conjunto de n pedidos $\{p_1, p_2, \ldots, p_n\}$

Cada pedido i tiene

un tiempo s_i donde inicia

Un tiempo t_i donde finaliza

un valor v_i

Un par de tareas $p_x, p_y \in P$

Son compatibles entre si, si - y solo si – no hay solapamiento en el tiempo entre ellas

Queremos

Seleccionar el subconjunto P con tareas compatibles entre si y que con la suma de sus valores lo mayor posible

¿Existe un algoritmo greedy para resolverlo?

Para el caso particular

$V_i = 1$ para todo i

Se corresponde

al problema de maximizar la cantidad de tareas compatibles a realizar

En ese caso

Funcionaba una estrategia greedy

Sin embargo

No se conoce una para el problema general

Un orden inicial

¿Qué hacíamos en la solución greedy?

Ordenábamos en orden creciente en tiempo de finalización de la tarea

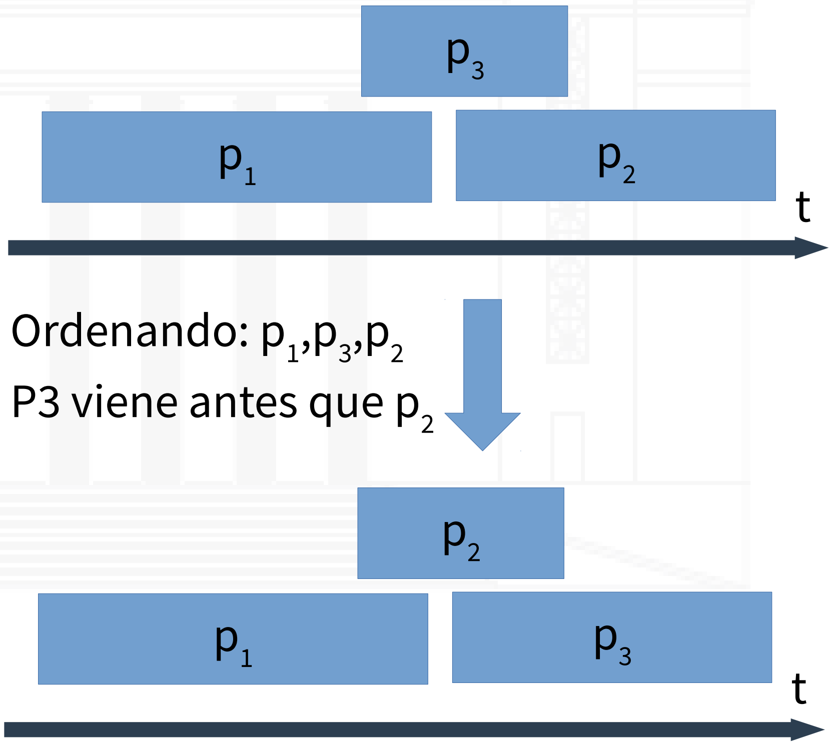
$f_1 \leq f_2 \leq \ldots \leq f_n$

Con este orden, podemos decir

Que la tarea i viene antes que la j, si i<j

Para construir nuestra solución

Utilizaremos el mismo ordenamiento



Tareas compatibles anteriores

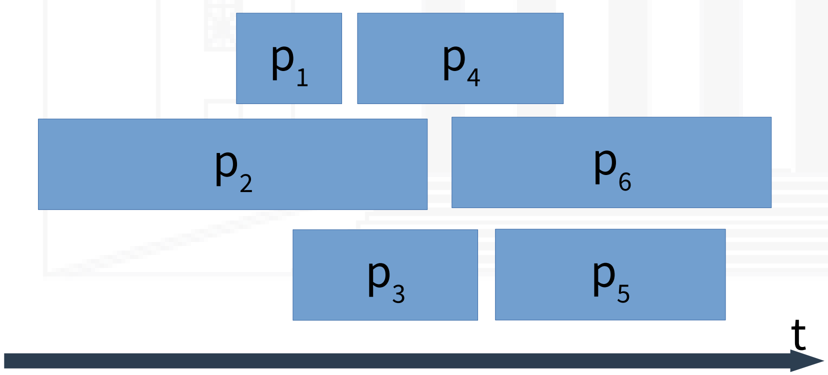
Para cada tarea i

Nos interesará conocer la primera tarea anterior con la que es compatible P(i)

Si nomenclamos las tareas utilizando el orden establecido

el indice de la tarea anterior compatible sera menor a la tarea → P(i)=x con x<i

Y todas las tareas con indice menor a x también serán compatibles



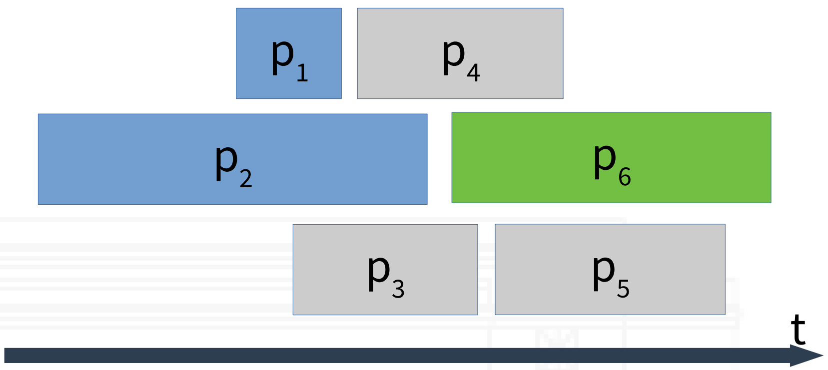
Tenemos:

P(6)=2 (y por lo tanto la tarea 1 también es compatible con la tarea 6)

P(5)= 3

P(3) = 0 (no hay ninguna tarea anterior compatible)

Pertenencia de una tarea al óptimo



Dada

Una instancia del problema

En la solución óptima O

Podrá pertenecer o no una determinada tarea i

Si pertenece

Entonces no podrán pertenecer aquellas incompatibles con ella

Recién P(i) podría pertenecer a la solución optima

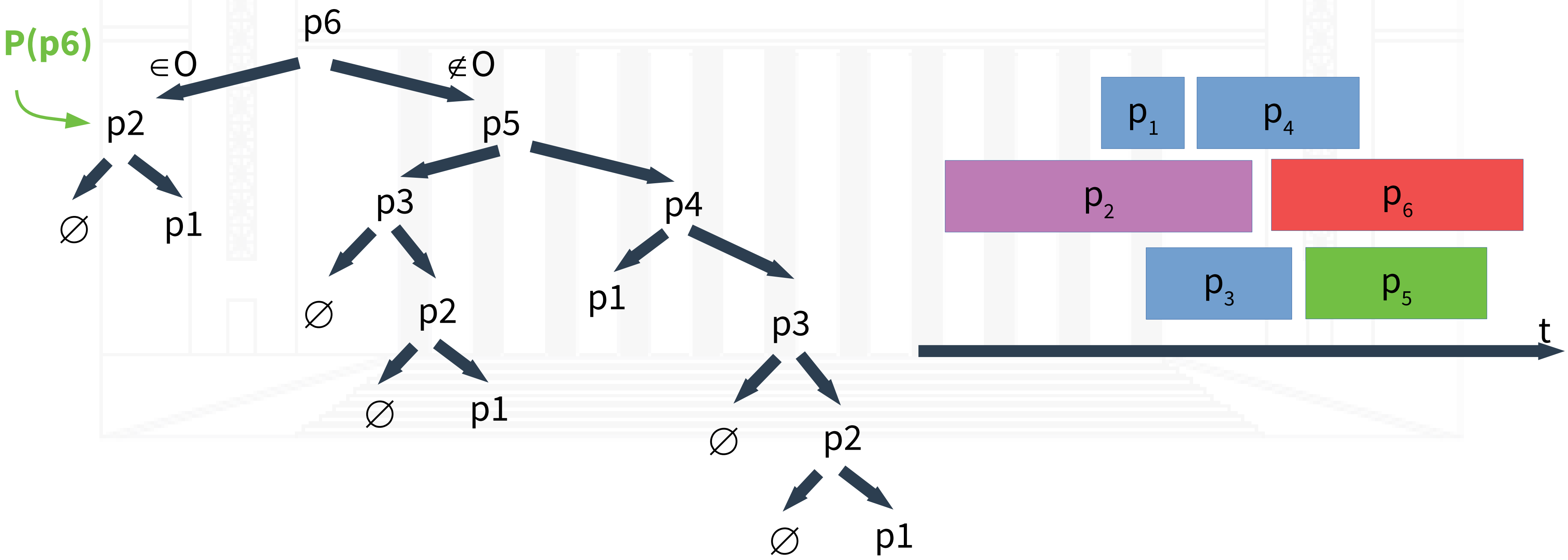
Si no pertenece

Entonces la tarea i-1 podría pertenecer a la solución optima

Árbol de decisión

Podemos utilizar este criterio

Comenzando por la tarea n y descendiendo hasta la tarea 1

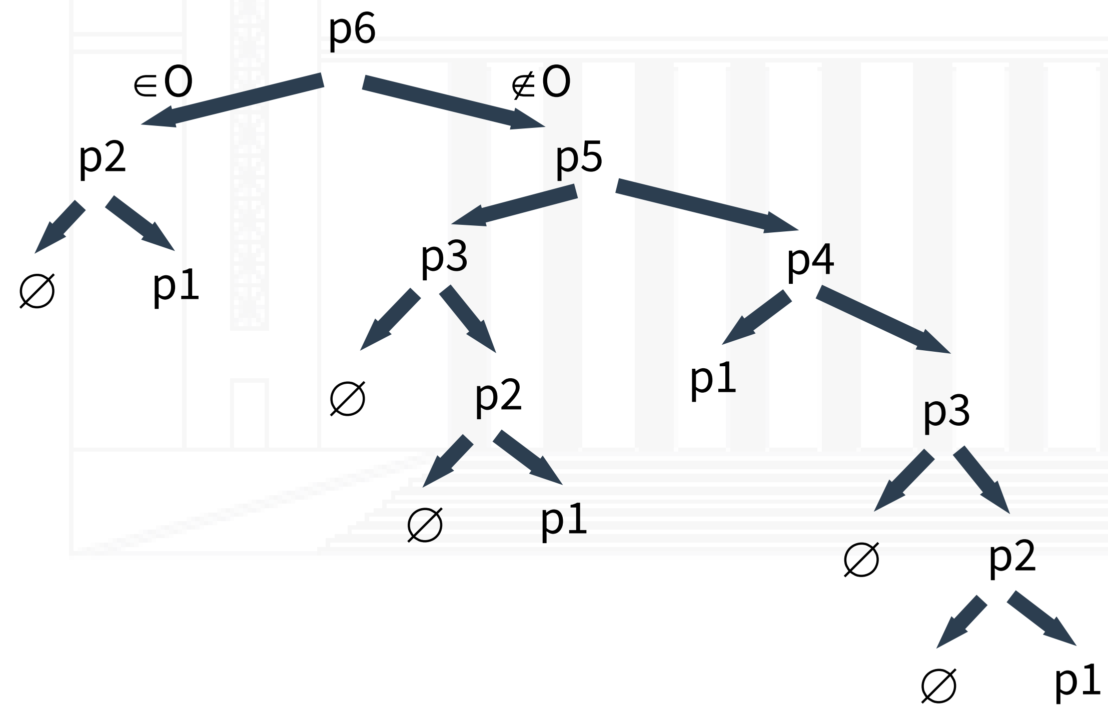


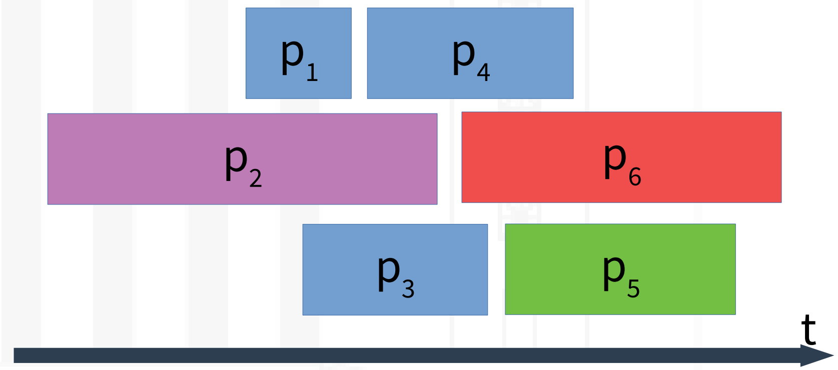
Tareas que pertenecen al óptimo (II)

¿Qué determina que una tarea i este o no en el óptimo?

Si conocemos el optimo de sus subproblemas: O(i-1) y O(p(i))

Elegimos el mayor valor entre v(pi)+O(i-1) y O(p(i))



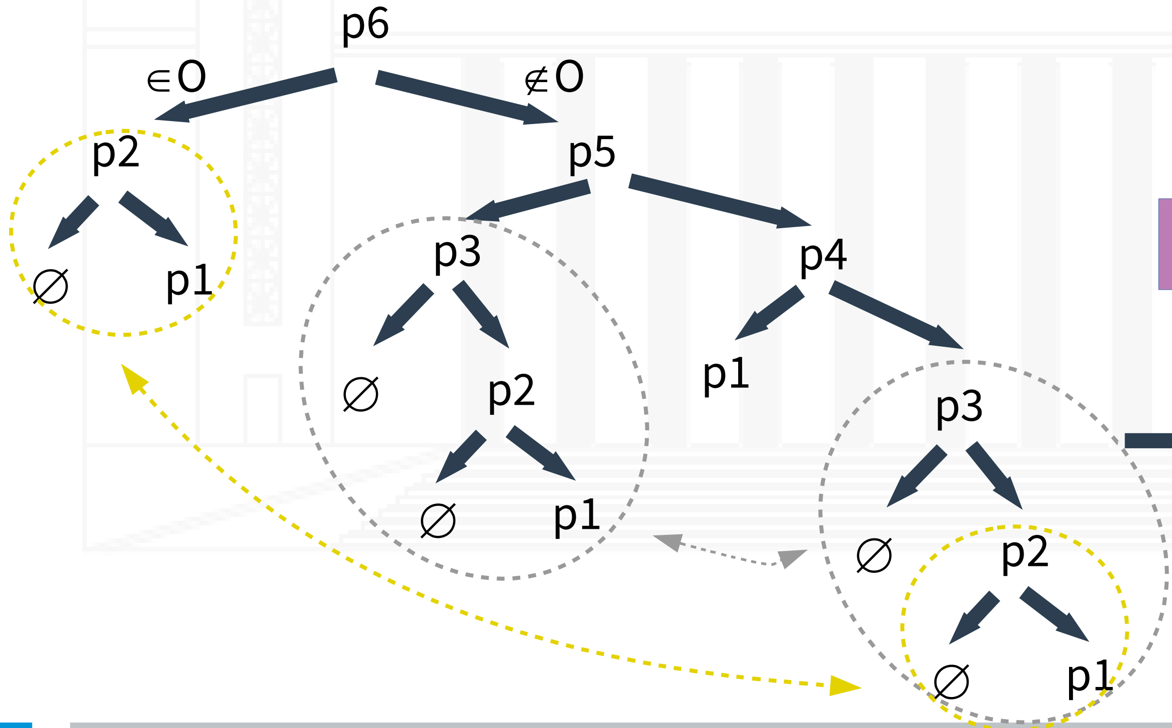


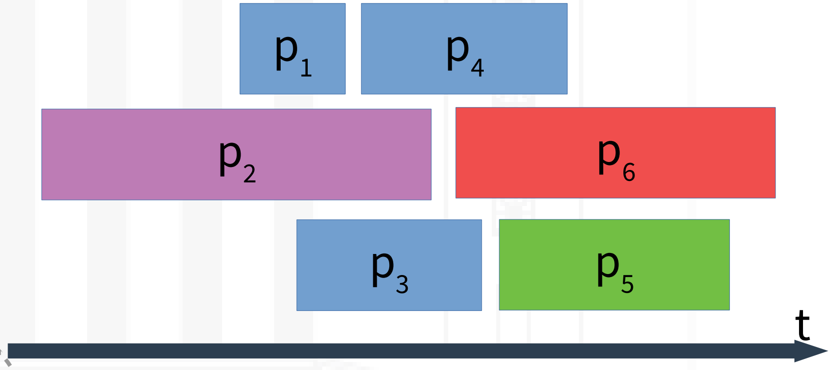
Memorización

¿Tenemos que recorrer (y calcular) todo el árbol?

Si observamos atentamente, algunos subproblemas se repiten

Alcanza con calcularlos solo 1 vez (aplicar memorización)





Recurrencia

Podemos expresar el problema como:

$$\begin{cases} OPT(x) = 0 & , si\ x = 0 \\ OPT(x) = max\{V(x) + OPT(P(x)), OPT(x-1)\} & , si\ X > 0 \end{cases}$$

El resultado con el máximo valor posibles será:

OPT(n)

Solución iterativa

Complejidad

Temporal O(n)

Espacial: O(n)

```
OPT[0]=0

Desde i=1 a n

    enOptimo = V[i] + OPT[P(i)]
    noEnOptimo = OPT[i-1]

    si enOptimo >= noEnOptimo
        OPT[i] = enOptimo
    sino
        OPT[i] = noEnOptimo

Retornar OPT[n]
```

Reconstruir las elecciones

Para cada subproblema i

almacenar si la tarea se eligió

Reconstruir para atrás

Partiendo de la tarea n

Iterar pasando por lo subproblemas seleccionados

```
OPT[0]=0
elegido[0]=false
Desde i=1 a n

    enOptimo = V[i] + OPT[P(i)]
    noEnOptimo = OPT[i-1]

    elegido[i]=(enOptimo >= noEnOptimo)
    si enOptimo >= noEnOptimo
        OPT[i] = enOptimo
    sino
        OPT[i] = noEnOptimo

Imprimir OPT[n]

i = n
Mientras i >0
    Si elegido[i]
        Imprimir i
        i = P(i)
    sino
        i--
```

Presentación realizada en Octubre de 2020