

# NOTE BOOK

Rokshana Ahmed

# Algorithms

DATE: 26/4/22

## ( LOCAL SEARCH ALGORITHMS )

- GREEDY ALGORITHMS
- DIVIDE - ET - IMPERA "

## WEIGHTED INTERNAL SCHEDULING

We are given a set of intervals :

$$I = \{i_1, \dots, i_n\} = \{(s_1, f_1), (s_2, f_2), \dots, (s_n, f_n)\}$$

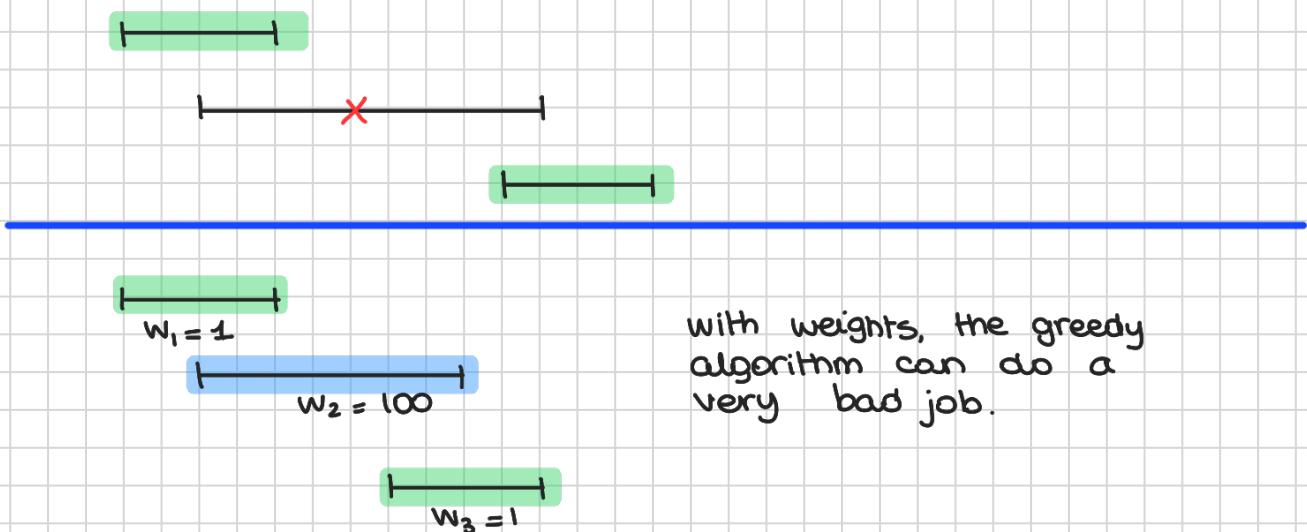
where  $s_i$  is the starting time of interval  $i$ , and  $f_i$  is the ending time of interval  $i$ , as well as a weighting function  $w$  that assigns a weight/value to each interval where  $w(i)$  is the weight/value of  $i$ .

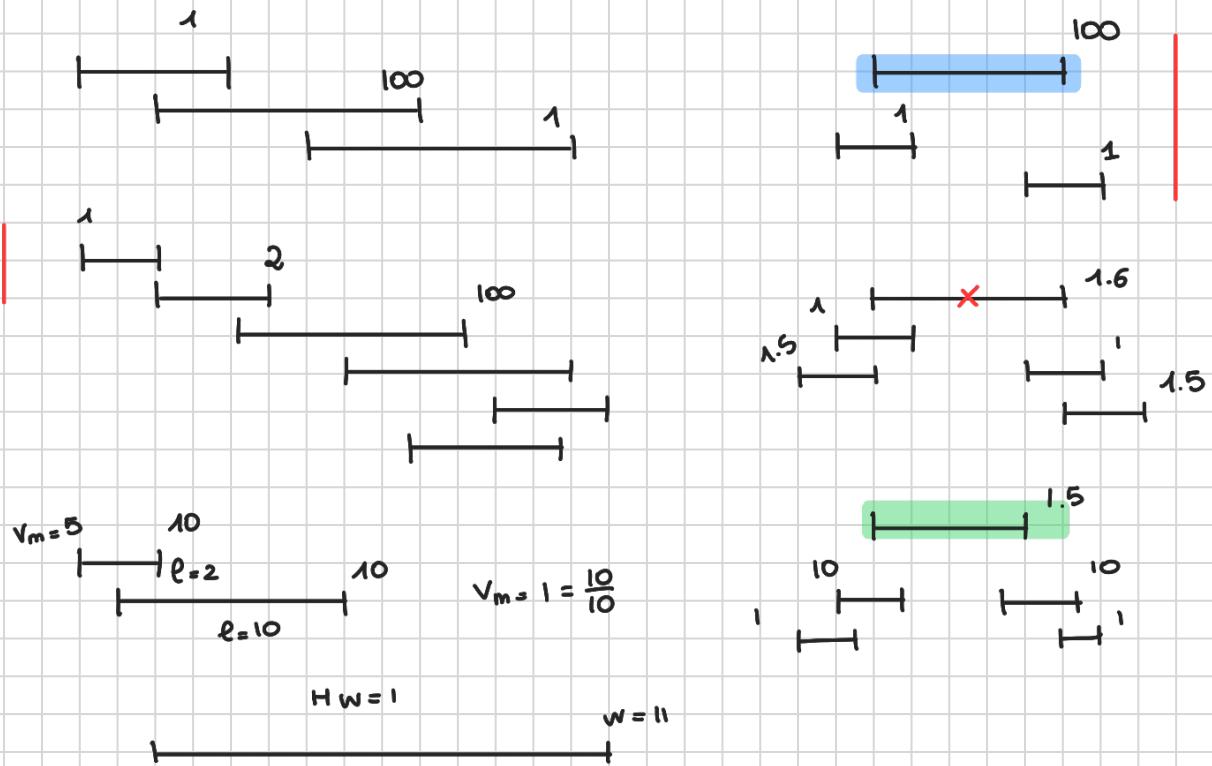
GOAL : Select a subset  $S \subseteq I$  of non-overlapping intervals of maximum value, that is, that maximises  $\sum_{i \in S} w(i)$

Intervals scheduling is a **special case** of weighted interval scheduling ( just set  $w(i) = 1 \forall i \in I$  ).

(Weighted interval sch. is a **generalization** of int. sch.)

We solved Interval scheduling with the "earliest-finishing-time" greedy algorithm.





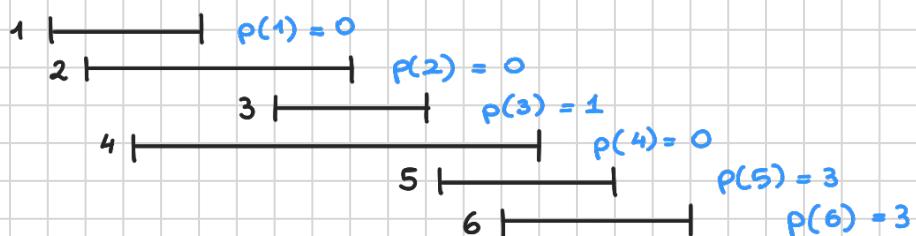
## DINAMYC PROGRAMMING

An algorithmic technique that keeps track of all solutions at once, in "little" time.  
 (Greedy algorithms, instead, consider one partial solution at a time).

First of all, let us assume that the intervals are sorted by finishing time :  $f_1 \leq f_2 \leq \dots \leq f_n$ .

**DEF:** INTERVAL i "comes before" interval j IFF  $f_i \leq f_j$ .

**DEF:** Let  $\rho(j)$ , for an interval j, be the largest index  $i < j$  S.T. intervals i and j are disjoint (compatible), or, if no such i exists, let  $\rho(j) = 0$ .



L: The generic interval  $i$  is disjoint (compatible) with each of the intervals  $1, 2, \dots, p(i)$ .

P:  $i$  is disjoint from  $p(i)$  intervals are sorted increasingly by their finishing time.  
 $i$  being compatible with  $p(i) < i$  then means that  $s_i > f_{p(i)}$ . Also,  $f_{p(i)} \geq f_{p(i)-1} \geq \dots \geq f_i$ . Thus,  $s_i > f_j$  for each  $j \in \{1, 2, \dots, p(i)\}$ . ■

Suppose that  $O_i$  is an optimal solution of the problem restricted to the first  $i$  intervals  $(\{(s_1, f_1), (s_2, f_2), \dots, (s_i, f_i)\})$

Let  $\text{OPT}_i$  be the value of  $O_i$

For each  $j \geq 1$ :

- either interval  $j$  is in  $O_j$ ,  $j \in O_j$ , then

$$\text{OPT}_j = w_j + \text{OPT}_{p(j)}.$$

- or  $j \notin O_j$ , then

$$\text{OPT}_j = \text{OPT}_{j-1}$$

OBS: ① Interval  $j$  is in an optimal solution  $O_j$  if

$$w_j + \text{OPT}_{p(j)} \geq \text{OPT}_{j-1}.$$

② Interval  $j$  is not in an optimal solution  $O_j$  if

$$\text{OPT}_{j-1} \geq w_j + \text{OPT}_{p(j)}.$$

Dynamic Programming, in general, expresses the value of optimal solutions in terms of the value of optimal solutions to smaller problems.

DEF Compute -  $\text{OPT}(j)$ :

// compute the value of  $\text{OPT}_j$  for  $j=0, 1, \dots, n$

IF  $j == 0$ :

RETURN 0

ELSE :

RETURN  $\max(w_j + \text{COMPUTE-OPT}(p(j)), \text{COMPUTE-OPT}(j-1))$

---

L: COMPUTE-OPT(j) returns  $\text{OPT}_j$  for  $j=0, 1, \dots, n$

P: BASE CASE:  $j=0$  and  $\text{OPT}(0) = 0 = \text{COMPUTE-OPT}(0)$ . ✓

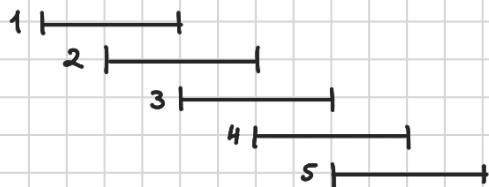
IND. STEP : Suppose that  $\text{COMPUTE-OPT}(i) = \text{OPT}(i)$   $\forall i \leq j$ .

Then, by OBS.,  $\text{OPT}(j+1) = \max(w_{j+1} + \text{OPT}(p(j+1)), \text{OPT}(j))$ .

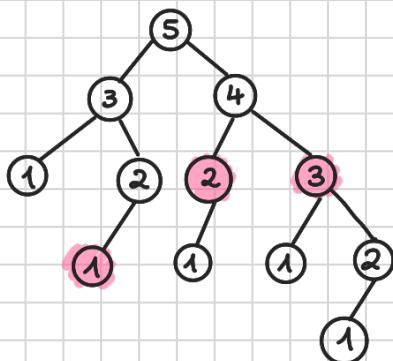
Then,  $\text{COMPUTE-OPT}(j+1) = \text{OPT}(j+1)$ . ■

---

The issue with the algorithm is that it takes exponential time



COMPUTE-OPT(5)



Then,  $\text{COMPUTE-OPT}(n)$  takes time  $\geq 2^{\frac{n}{2}}$

---

M - COMPUTE-OPT(j) :

MEMOIZATION

GLOBAL M



IF j A KEY OF M :

RETURN M[j]

ELSE :

IF  $j == 0$  :

$M[j] = 0$

ELSE :

$M[j] = \max (w_j + M - \text{COMPUTE-OPT}(p(j)), M - \text{COMPUTE-OPT}(j-1))$

RETURN  $M[j]$

L :  $M\text{-COMPUTE-OPT}(j)$  RETURNS  $\text{OPT}_j = \text{COMPUTE-OPT}(j)$

L :  $M\text{-COMPUTE-OPT}(j)$  TAKES  $O(j)$  TIME.

EXERCISE :

Write down a version of  $M\text{-COMPUTE-COST}$  that is not recursive.