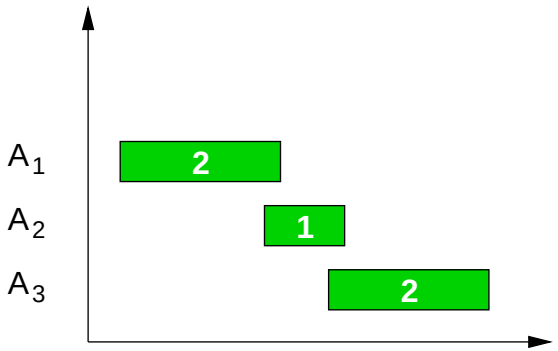
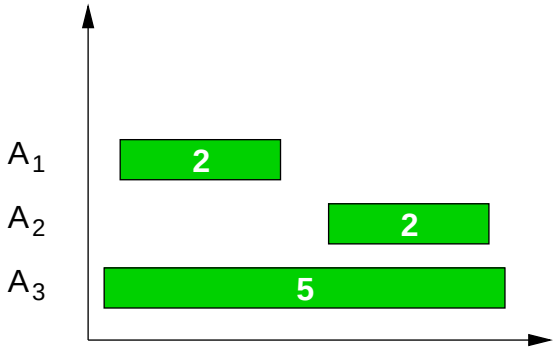
Greedy: is it all as simple as that?

Weighted Activity-selection problem. We are given a set of proposed activities $\mathcal{S} = \{A_1, A_2, \ldots, A_n\}$ that wish to use a resource, which can be used by only one activity at a time. Each activity is defined by a *start time* s_i, a *finish time* f_i, with $0 \leq s_i < f_i < +\infty$, and a positive integer value w_i. If selected, activity A_i takes place during the time interval $[s_i, f_i)$. Two activities A_i and A_j are *compatible* if $s_i \geq f_j$ or $s_j \geq f_i$. The *weighted* activity-selection problem is to select a number of mutually compatible activities of the largest possible total weight.

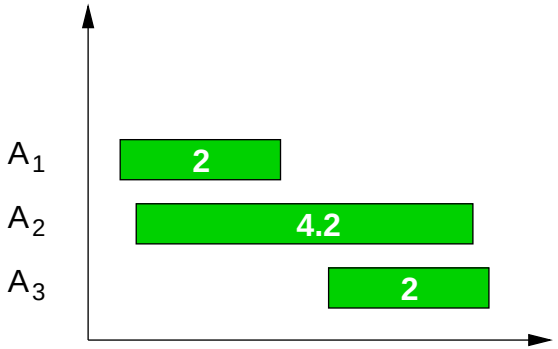
Example



One more example



... and again



Remarks

No (optimal) greedy algorithm is known for the Weighted Activity Selection problem.

The "Greedy-choice" property seems to fail! An activity that ends the earliest is not guaranteed to be the most profitable.

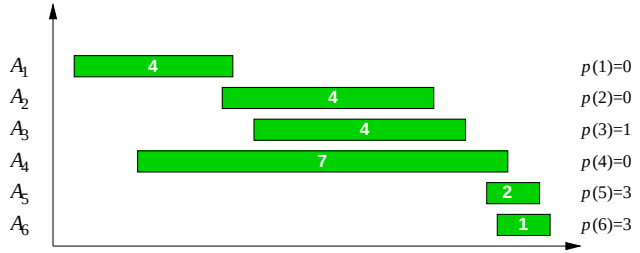
What can be done??

Go for Divide and Conquer!

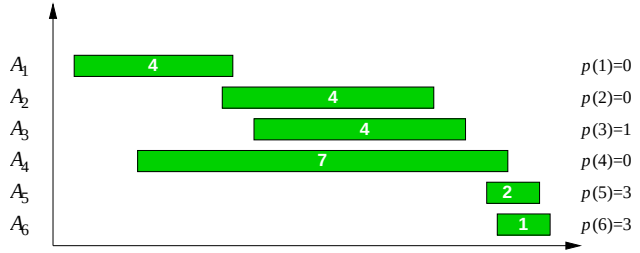
We assume the activities are sorted by nondecreasing finish time.

For each j define $p(j)$ to be the largest index smaller than j of an activity compatible with A_j ($p(j) = 0$ if such index does not exist). In the last example $p(1) = p(2) = 0$, and $p(3) = 1$.

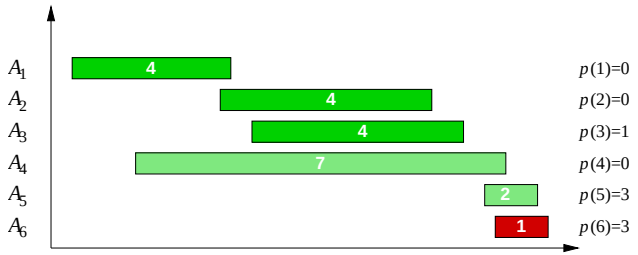
Look at an optimal solution



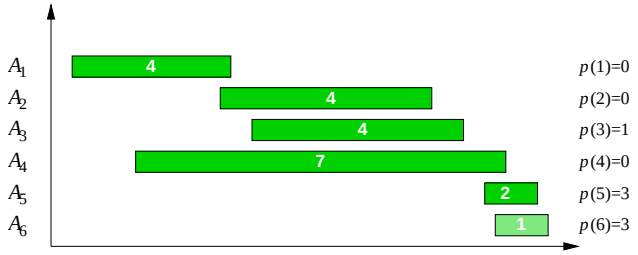
Look at an optimal solution



Let X be an optimal solution. The last activity A_n can either be in X or not.



Case 1. A_n in X: then X cannot contain any other activity with index larger than $p(n)$.



Case 2. A_n is not in X. We can throw it away and consider $A_1, \ldots, A_{n-1}$.

To summarize either the optimal solution is formed by some solution to the instance formed by activities $A_1, \ldots, A_{p(n)}$ plus A_n or the optimal solution does not contain A_n (and therefore it is some solution to the instance formed by activities $A_1, \ldots, A_{n-1}$). Of course to maximize the size of the solution to the whole instance, we'd better choose optimal solutions to $A_1, \ldots, A_{p(n)}$ or $A_1, \ldots, A_{n-1}$ (in other words the problem has the optimal sub-problem property).

In other words:

$$\text{OPT}(n) = \max\{A_n + \text{OPT}(p(n)), \text{OPT}(n-1)\},$$

and

$$\text{cost(OPT}(n)) = \max\{w_n + \text{cost(OPT}(p(n))), \text{cost(OPT}(n-1))\}.$$

Recursive solution

Based on the argument so far we can at least write a recursive method that computes the cost of an optimal solution ... rather inefficiently. The list of all weights is a global variable $(w_1, \ldots, w_n)$ to

```
RECURSIVE-WEIGHTED-SELECTOR (j)
  if j = 0
    return 0
  else
    return max{w_j + RECURSIVE-WEIGHTED-SELECTOR (p(j)),
           RECURSIVE-WEIGHTED-SELECTOR (j − 1)}
```

The optimum for the given instance on n activities is obtained by running RECURSIVE-WEIGHTED-SELECTOR (n).

Memoization

The array $M[j]$ (another global variable) contains the size of the optima for the instances $A_1, \ldots, A_j$.

```
FAST-RECURSIVE-WEIGHTED-SELECTOR (j)
  if j = 0
    return 0
  else if M[j] not empty
    return M[j]
  else
    M[j] ← max{w_j + FAST-RECURSIVE-WEIGHTED-SELECTOR (p(j)),
          FAST-RECURSIVE-WEIGHTED-SELECTOR (j − 1)}
    return M[j]
```


Exercises

1. Derive a recursive algorithm that actually computes the optimal set of activities from the algorithm RECURSIVE-WEIGHTED-SELECTOR.
2. Derive a recursive algorithm that actually computes the optimal set of activities from the algorithm FAST-RECURSIVE-WEIGHTED-SELECTOR.
3. Simulate both RECURSIVE-WEIGHTED-SELECTOR and FAST-RECURSIVE-WEIGHTED-SELECTOR on the following instance:

	A_1	A_2	A_3	A_4	A_5
s	1	2	3	5	3
f	3	6	7	7	10
w	4	2	4	2	6