11 Weighted Interval Scheduling [KT 6.1, 6.2]

Greedy doesn't work. See Fig. 6.1 for example.

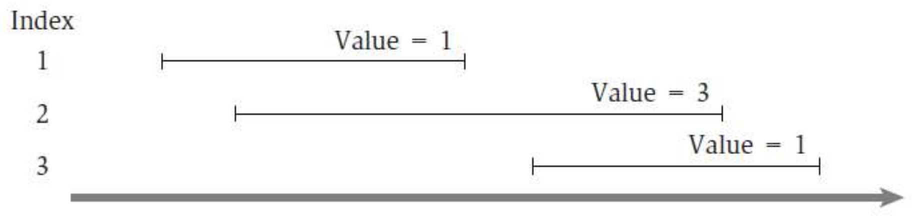

Figure 6.1 A simple instance of weighted interval scheduling.

DP formulation. Sort by finishing time. Define $p(i)$ be the predecessor of i. See Fig. 6.2 for example.

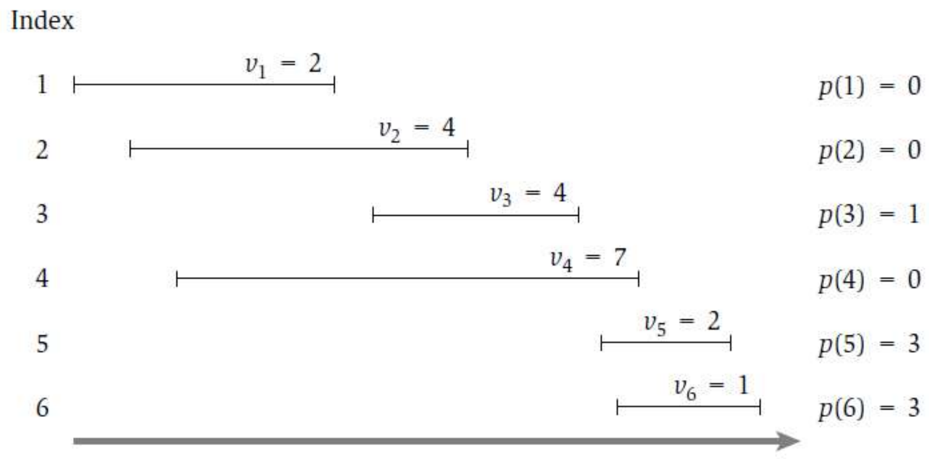

Figure 6.2 An instance of weighted interval scheduling with the functions $p(j)$ defined for each interval j.

The DP can be described by the following recursion:

$$OPT(j) = \max(v_j + OPT(p(j)), OPT(j-1)).$$

Clever brute-force computation. Direct implementation using recursion gives exponential running time.

```
Compute-Opt($j$)
  If $j=0$ then
    Return 0
  Else
    Return max($v_j$+Compute-Opt(p(j)), Compute-Opt($j-1$))
  Endif
```

The recursion will be unrolled as follows:

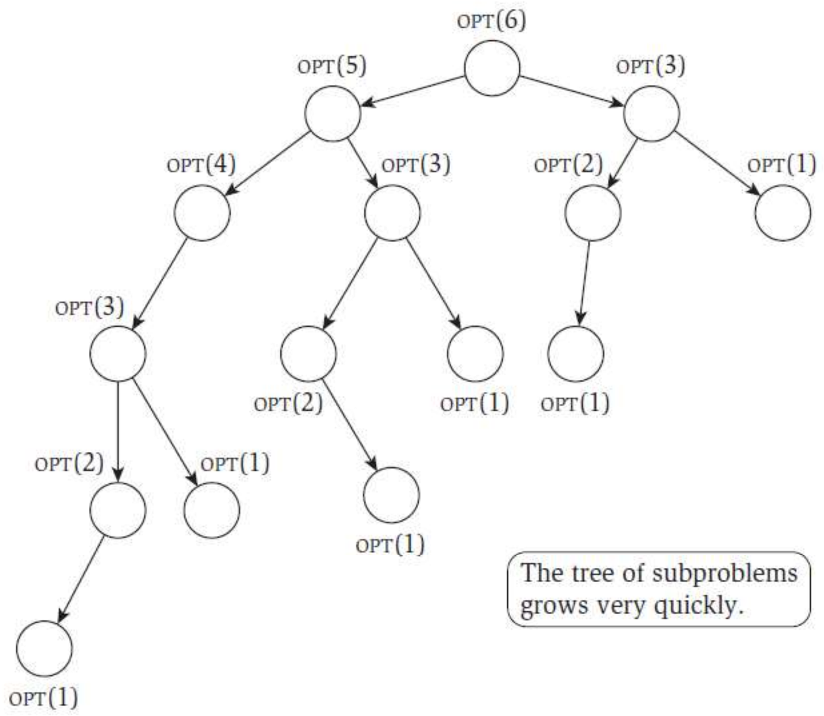

Figure 6.3 The tree of subproblems called by Compute-Opt on the problem instance of Figure 6.2.

Instead, we try to memorize the already compute values in the recursion, as illustrated in the following algorithm.

```
Iterative-Compute-Opt
  $M[0]=0$
  For $j=1,2,\ldots,n$
    $M[j]=\max(v_j+M[p(j)], M[j-1])$
  Endfor
```

Running time. $O(n)$ if input intervals are sorted according to their finishing time, and $p(i)$'s are known.

How to compute $p(i)$'s? Sort by finishing time, getting list A. Sort by starting time, getting list B. Scan lists A and B at the same time (time $O(n)$). So the final total running time is dominated by sorting.

Computing a solution. We can go over the array $M[1..n]$ again to compute an actual solution.

```
Find-Solution($j$)
  If $j = 0$ then
    Output nothing
  Else
    If $v_j + M[p(j)] \geq M[j-1]$ then
      Output $j$ together with the result of Find-Solution($p(j)$)
    Else
      Output the result of Find-Solution($j-1$)
    Endif
  Endif
```

General guideline of DP.

1. Iteratively build up of subproblems
2. Make sure that there are polynomial number of subproblems.
3. The solution of the original problem can be easily computed from solutions to the subproblems
4. The solution to each subproblem can be determined from a small number of "smaller" subproblems.

I think the key is to formulate subproblems – often needs to understand the structure of the problem, or have some idea of *solving by"recursion"* in mind. The is **the art of algorithm design**.