1.5em 0pt

Dynamic Programming

Dr. Raghunath Reddy M

Dept. of Computer Science Engineering,
Bennett University, Greater Noida

February 12, 2020

Weighted Interval Selection / Activity Scheduling Problem

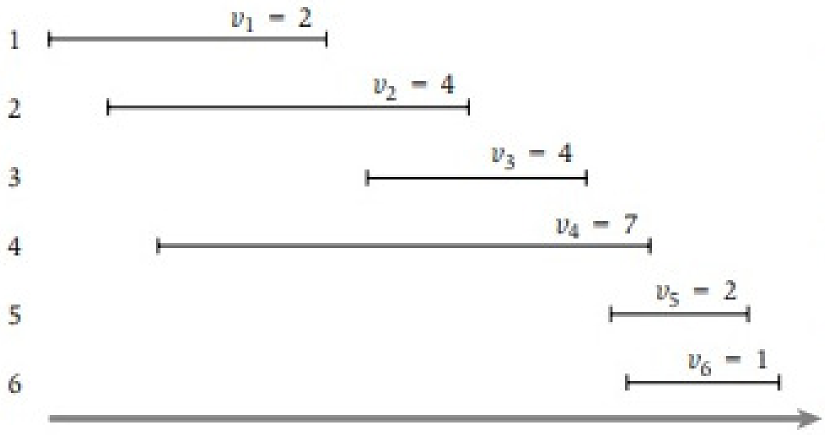


Figure: An instance of the problem.

Weighted Interval Selection / Activity Scheduling Problem

Problem Definition

Input: We are given a set of n intervals labeled $1, 2, \ldots, n$ and each interval i is of the form (s_i, f_i) $(s_i < f_i)$. Further, every interval i has a value (profit) v_i.

Output: Find a subset S of $\{1, 2, \ldots, n\}$ such that

1. No two intervals in S overlap and
2. $\sum_{i \in S} v_i$ is maximum.

Question

Any of the greedy selection rules:

1. least (minimum) finishing time (f_i)
2. Pick the maximum value interval / activity

give optimal solution ?

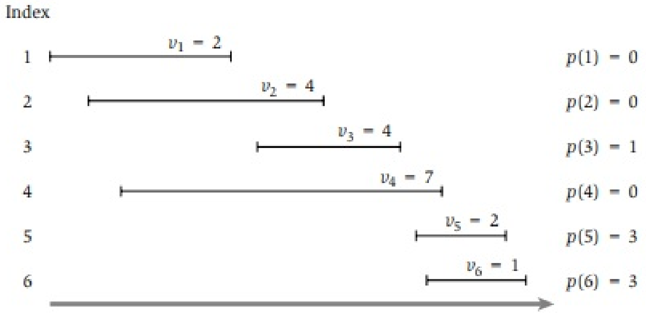
Ans: NO

Dynamic Programming

- Let's suppose that the intervals are sorted in order of non-decreasing finish time: $f_1 \leq f_2 \leq \cdots \leq f_n$.

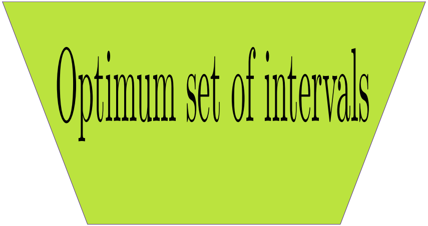
Notation

For any given interval j, let $p(j)$ be the largest index $i < j$ such that both intervals i and j do not overlap.



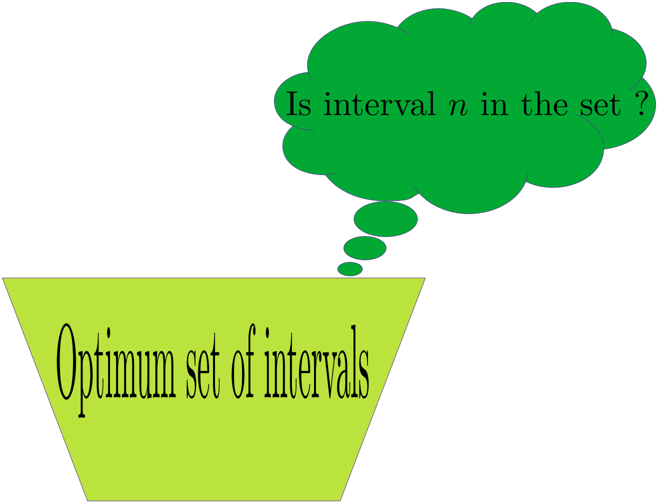
Some observation about an optimum solution

- Let $\mathcal{O}$ be an optimal set of intervals (which are disjoint and has maximum total value)
- Actually, we do not know $\mathcal{O}$. In fact, we have to compute $\mathcal{O}$.



Set $\mathcal{O}$

Look for interval n in the set $\mathcal{O}$



Set $\mathcal{O}$

Case (i): $n \notin \mathcal{O}$

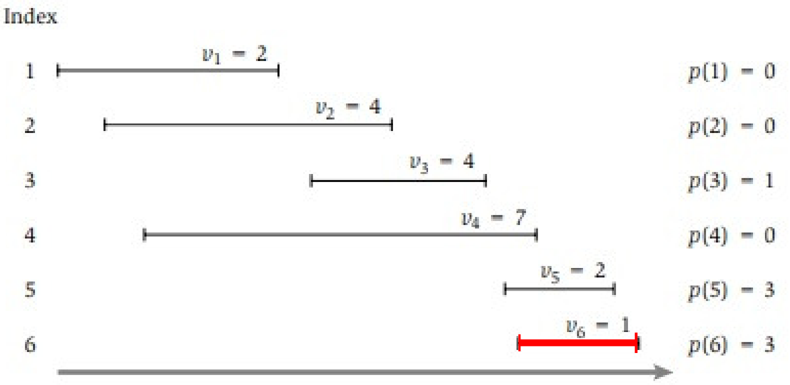
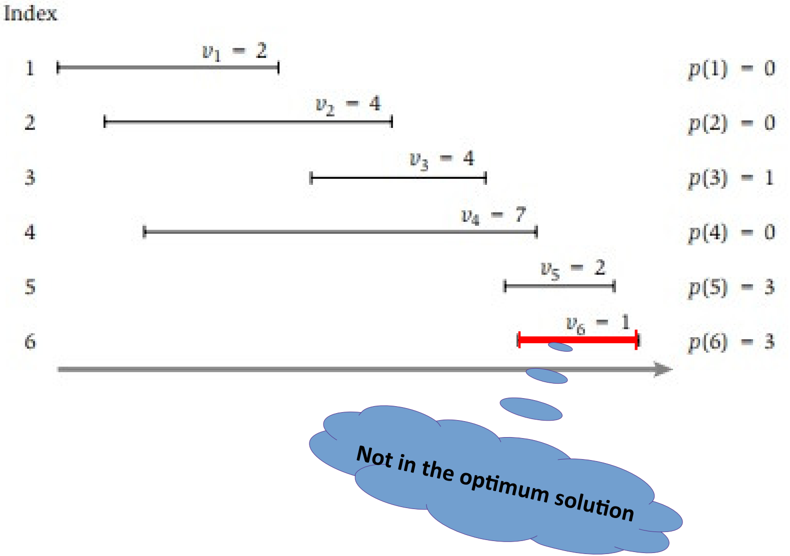
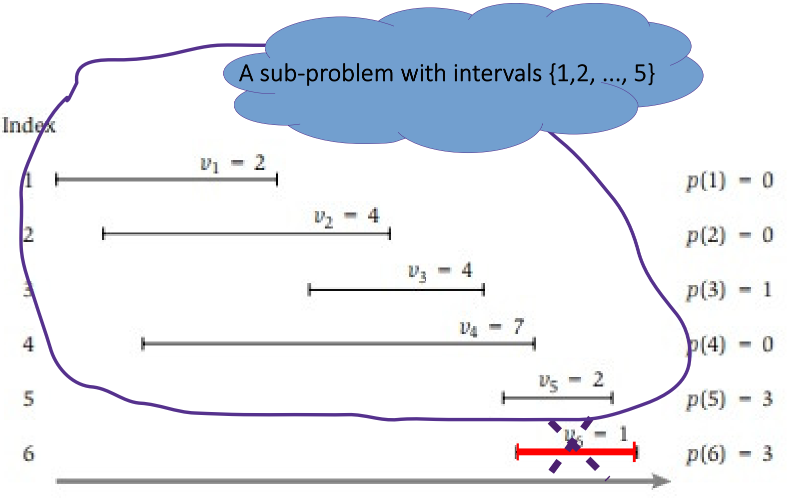


Figure: Consider an instance with intervals $\{1, 2, 3, 4, 5, 6\}$.

Case (i): $n \notin \mathcal{O}$

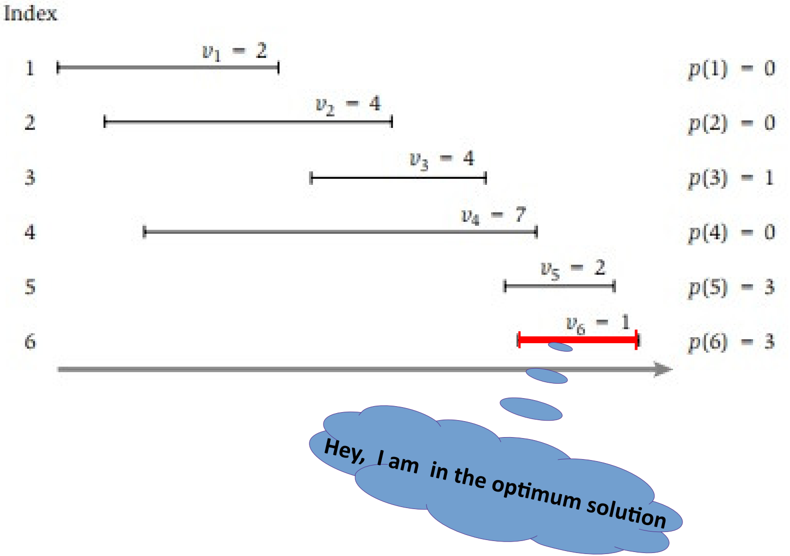


Case (i): $n \notin \mathcal{O}$

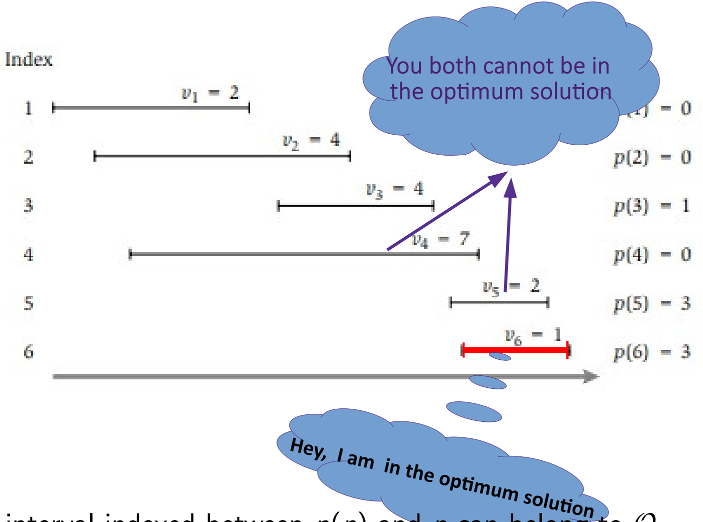


- $\mathcal{O}$ is the same as the optimal solution to the problem consists of internals $\{1, 2, \ldots, n-1\}$.

Case (ii): $n \in \mathcal{O}$

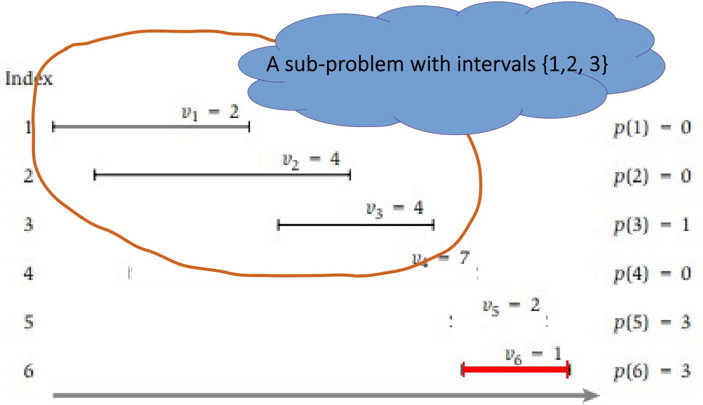


Case (ii): $n \in \mathcal{O}$



- No interval indexed between $p(n)$ and n can belong to $\mathcal{O}$.

Case (ii): $n \in \mathcal{O}$



- Further, $\mathcal{O}$ must include an optimum solution to the problem consists of intervals $\{1, 2, \ldots, p(n)\}$ i.e., $\mathcal{O} = \{n\} \;\cup$ the optimal solution of the problem with intervals $\{1, 2, \ldots, p(n)\}$.

Connecting the dots

Case (i): $n \notin \mathcal{O}$

$\mathcal{O}$ is the same as the optimal solution to the problem consists of internals $\{1, 2, \ldots, n-1\}$.

Case (ii): $n \in \mathcal{O}$

$\mathcal{O} = \{n\} \;\cup$ the optimal solution of the problem with intervals $\{1, 2, \ldots, p(n)\}$.

Combining the both cases

Let $\mathcal{O}_j$ be the optimum set of intervals for the sub-problem with intervals $\{1, 2, \ldots, j\}$. Then,

- $\mathcal{O} = \mathcal{O}_n$ and
- further, $\mathcal{O}$ is either $\{n\} \cup \mathcal{O}_{p(n)}$ or $\mathcal{O}_{n-1}$.

Generalization of the facts

1. Let $\mathcal{O}_j$ be the optimum set of intervals for the sub-problem with intervals $\{1, 2, \ldots, j\}$.
2. Further, let $OPT(j)$ be the value of the solution $\mathcal{O}_j$ i.e., the sum of values of intervals in $\mathcal{O}_j$.

Case (i): $j \in \mathcal{O}_j$

1. $\mathcal{O}_j = j \cup \mathcal{O}_{p(j)}$.
2. $OPT(j) = v_j + OPT(p(j)$

Case (ii): $j \notin \mathcal{O}_j$

- $\mathcal{O}_j = \mathcal{O}_{j-1}$ and
- $OPT(j) = OPT(j-1)$

Connecting the dots

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

Algorithm

- Recall that $OPT(j) = \max\{v_j + OPT(p(j)), OPT(j-1)\}$

Algo. to compute $OPT(j)$ for a given j

$ComputeOpt(j)$

- If $j = 0$ then
  - return 0
- Else
  - return $\max\{v_j + ComputeOpt(p(j)), ComputeOpt(j-1)\}$

Lemma 0.1.

$ComputeOpt(j)$ correctly computes $OPT(j)$ for each $j = 1, 2, \ldots, n$.

Remark

$ComputeOpt(j)$ takes $O(2^j)$-time to return the value.

Reason for exponential time

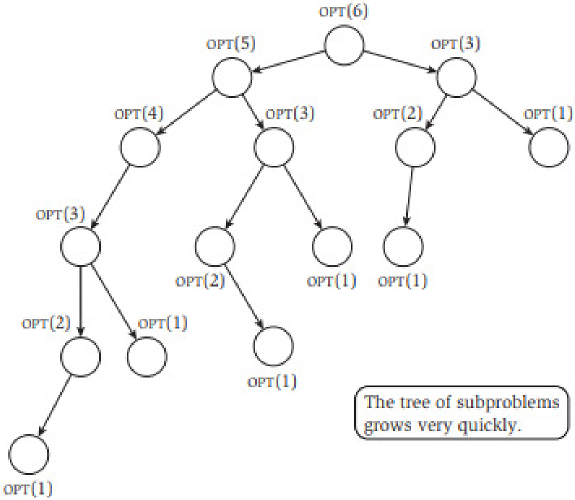


Figure: $OPT(6)$ computation tree

Reason for exponential time

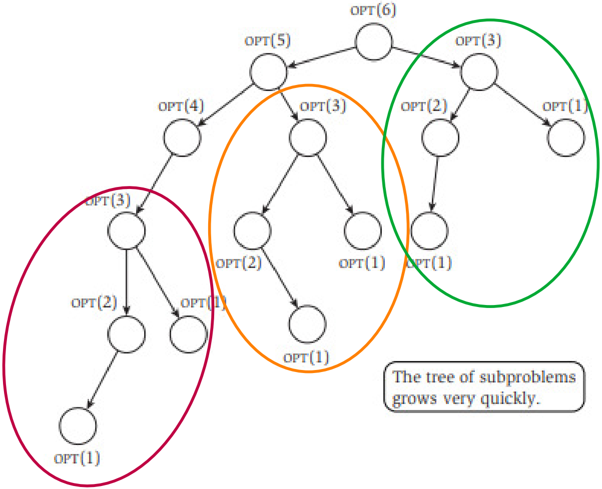


Figure: $OPT(6)$ computation tree

Memorizing the recursion

- Let $M[0..n]$ be a global array of size $n+1$ and initially, all the locations are empty.

Modifed Algo. to compute $OPT(j)$ for a given j

$M - ComputeOpt(j)$

- If $j = 0$ then
  - return 0
- Else if $M[j]$ is not empty then
  - return $M[j]$
- Else
  - $M[j] \;=\; \max\{\, v_j + ComputeOpt(p(j)),\ ComputeOpt(j-1)\}$
  - return $M[j]$

Lemma 0.2.

The running time of the above algorithm is $O(n)$.

Find the set of intervals using array M

- Recall that $OPT(j) = \max\{v_j + OPT(p(j)), OPT(j-1)\}$
- Interval j belongs to an optimum solution for the set of intervals $\{1, 2, \ldots, j\}$ if and only if $v_j + OPT(p(j)) \geq OPT(j-1)$.

Algo.

FindSolution(j)

- If $j = 0$ then
  - output nothing
- Else if $v_j + M[p(j)] \geq M[j-1]$
  - output j
  - call *FindSolution*$(p(j))$
- Else
  - call *FindSolution*$(j-1)$

Memorization vs Iteration over sub-problems

Algo. with memorization over sub-problems

$M - ComputeOpt(j)$

- If $j = 0$ then
  - return 0
- Else if $M[j]$ is not empty then
  - return $M[j]$
- Else
  - $M[j] \;=\; \max\{\, v_j + ComputeOpt(p(j)),\; ComputeOpt(j-1)\}$
  - return $M[j]$

Iteration over sub-problems

- $M[0] = 0$
- For $j = 1, 2, \ldots, n$
  - $M[j] = \max\{v_j + M[p(j)],\; M[j-1]\}$
- EndFor