EE360C ALGORITHMS - FALL 2018 - Tuesday Nov. 6 - Evening

Let J = set of jobs given as input

$OPT(J)$ = value of the optimal solution

Look at a particular job j:

Recursive expression for OPT:

$$OPT(J) = \max \{ OPT(J \setminus \{j\}), \quad \text{// if } j \text{ not part of opt. solution}$$
$$OPT(J \setminus \{j \text{ and all jobs overlapping with } j\}) + v(j) \quad \text{// if } j \text{ is part of the optimal solution} \}$$

$OPT(\{\}) = 0$ // base case

How can we generate subproblems efficiently (in constant time).

Set of jobs $1, \dots, n$, ordered by finish time.

$OPT(j)$ is value of optimal solution to the problem consisting of jobs $1, \dots, j$.

integer

$$OPT(j) = \begin{cases} 0, & \text{if } j = 0 \\ \max\{ OPT(j-1), \; v_j + OPT(p(j)) \} \end{cases}$$

$p(j)$: largest index $p(j) < j$ s.t. job $p(j)$ is compatible with j.

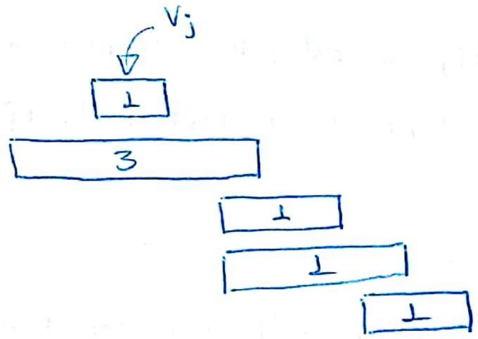
This is an arbitrary way to organise the jobs.
If we had sorted by start time, it would still be correct.

Running time

- Sort: $\Theta(n\log n)$
- Compute $p(j)$'s: $O(n\log n)$ //binary search for $p(j)$
- Iterative loop: $O(n)$

Overall: $O(n\log n)$

index



1 $\quad p(1)=0$
2 $\quad p(2)=0$
3 $\quad p(3)=1$
4 $\quad p(4)=1$
5 $\quad p(5)=3$

$$OPT(5) = \max\{\underbrace{OPT(4)}_{=3},\ 1+\underbrace{OPT(3)}_{=4}\} = 4$$
$$OPT(4) = \max\{\underbrace{OPT(3)}_{=3},\ 1+\underbrace{OPT(1)}_{=1}\} = 3$$
$$OPT(3) = \max\{\underbrace{OPT(2)}_{=3},\ 1+\underbrace{OPT(1)}_{=2}\} = 3$$
$$OPT(2) = \max\{\underbrace{OPT(1)}_{=1},\ 3+\underbrace{OPT(0)}_{=3}\} = 3$$
$$OPT(1) = \max\{\underbrace{OPT(0)}_{=0},\ 1+\underbrace{OPT(0)}_{=1}\} = 1$$

$M[j] := OPT(j)$

M =

0	1	2	3	4	5
0	1	3	3	3	4

OUTPUT : OPT(n)

Apply FIND-SOLUTION(n) algorithm from the slides:

Find-SOLUTION(5)
Find-SOLUTION(3) $\qquad$ OUTPUT: 5 , 2
Find-Solution(2)
Find-Solution(0)

KNAPSACK

Given n items, each with weight $w_i > 0$ & value v_i.
integer

Knapsack can hold at most W total weight.

Goal: Fill the knapsack so as to maximize total value.