

Lecture 29

CSE 331

Nov 6, 2019

My OH today at 2pm

note ☆ stop following 62 views

Atri Wednesday OH

Just for this week, I will have to move my Wed office hours to **2pm** instead of the usual 3pm since I need to attend something at 3pm related to department recruitment.

Apologies for any inconvenience this might cause.

#pin

office_hours

edit · good note | 0 Updated 21 hours ago by Atri Rudra

End of Semester blues

Can only do one thing at any day: what is the optimal schedule to obtain maximum value?



Write up a term paper (10)

Party! (2)

Exam study (5)

331 HW (3)

Project (30)

Monday

Tuesday

Wednesday

Thursday

Friday

Previous Greedy algorithm

Order by end time and pick jobs greedily

Greedy value = $5+2+3=10$

Write up a term paper (10)

Party! (2)

Exam study (5)

331 HW (3)

Project (30)

OPT = 30

Monday

Tuesday

Wednesday

Thursday

Friday



Weighted Interval Scheduling

Input: n jobs (s_i, f_i, v_i)

Output: A schedule S s.t. no two jobs in S have a conflict

Goal: $\max \sum_{i \in S} v_j$

Assume: jobs are sorted by their finish time

Today's agenda

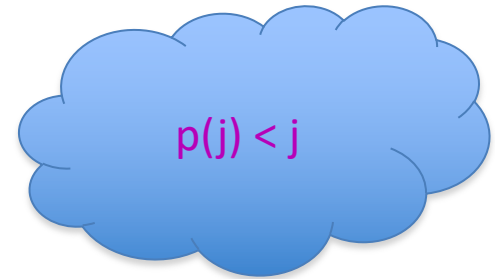
Finish designing a recursive algorithm for the problem



Couple more definitions

$p(j)$ = largest $i < j$ s.t. i does not conflict with j

= 0 if no such i exists



$OPT(j)$ = optimal value on instance $1, \dots, j$

Property of OPT

j in $\text{OPT}(j)$

j not in $\text{OPT}(j)$

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

Given $\text{OPT}(1), \dots, \text{OPT}(j-1)$,
how can one figure out if j
in optimal solution or not?



A recursive algorithm

Compute-Opt(j)

Correct for $j=0$

Proof of correctness by induction on j

If $j = 0$ then return 0

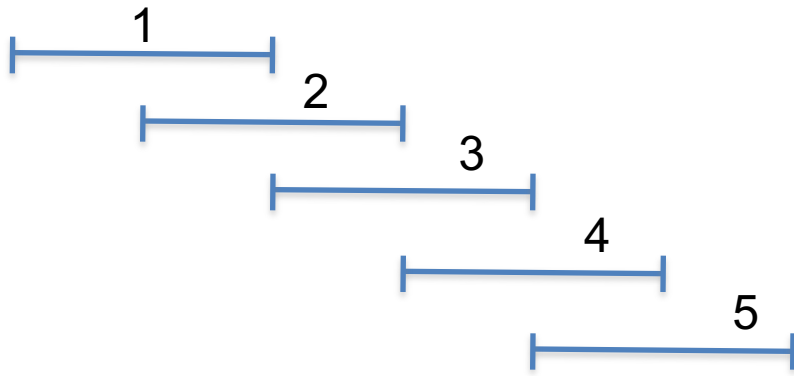
return $\max \{ v_j + \text{Compute-Opt}(p(j)), \text{Compute-Opt}(j-1) \}$

$= \text{OPT}(p(j))$

$= \text{OPT}(j-1)$

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

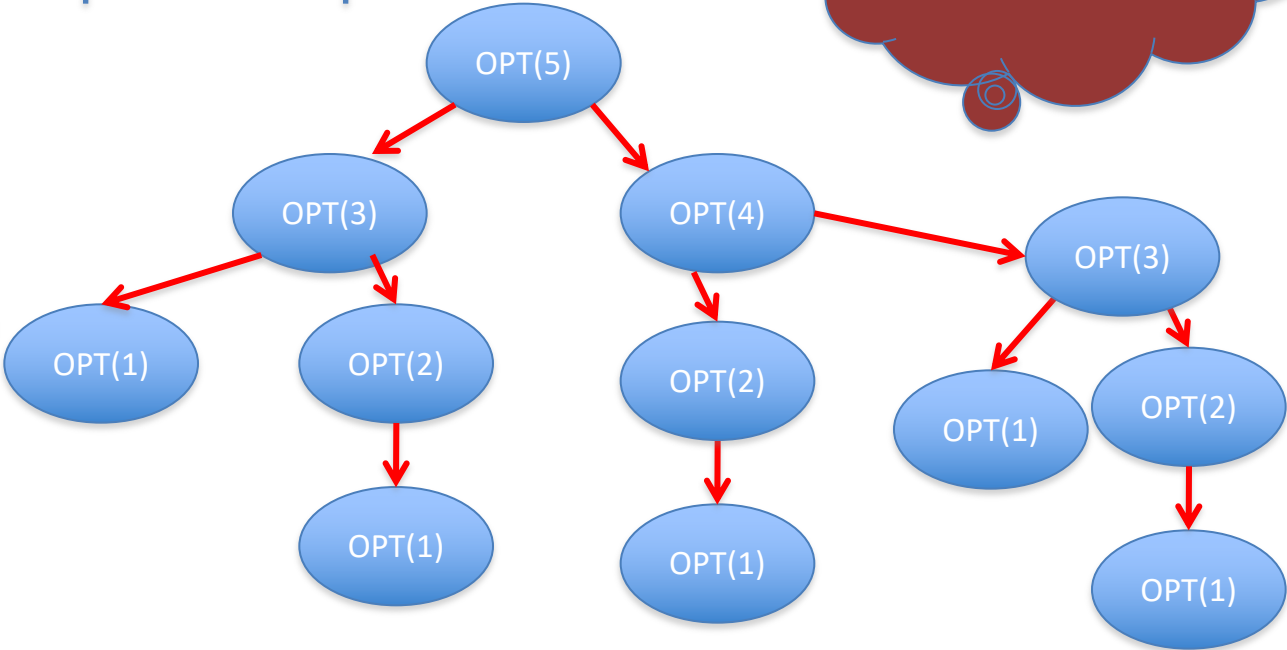
Exponential Running Time



$$p(j) = j - 2$$

Only 5 OPT values!

Formal proof: Ex.





Using Memory to be smarter

Using more space can reduce runtime!

How many distinct OPT values?

A recursive algorithm

M-Compute-Opt(j)

If $j = 0$ then return 0

If $M[j]$ is not null then return $M[j]$

$M[j] = \max \{ v_j + \text{M-Compute-Opt}(p(j)), \text{M-Compute-Opt}(j-1) \}$

return $M[j]$

M-Compute-Opt(j)
= OPT(j)

Run time = $O(\# \text{ recursive calls})$

Bounding # recursions

M-Compute-Opt(j)

If $j = 0$ then return 0

If $M[j]$ is not null then return $M[j]$

$M[j] = \max \{ v_j + M\text{-Compute-Opt}(p(j)), M\text{-Compute-Opt}(j-1) \}$

return $M[j]$

$O(n)$ overall

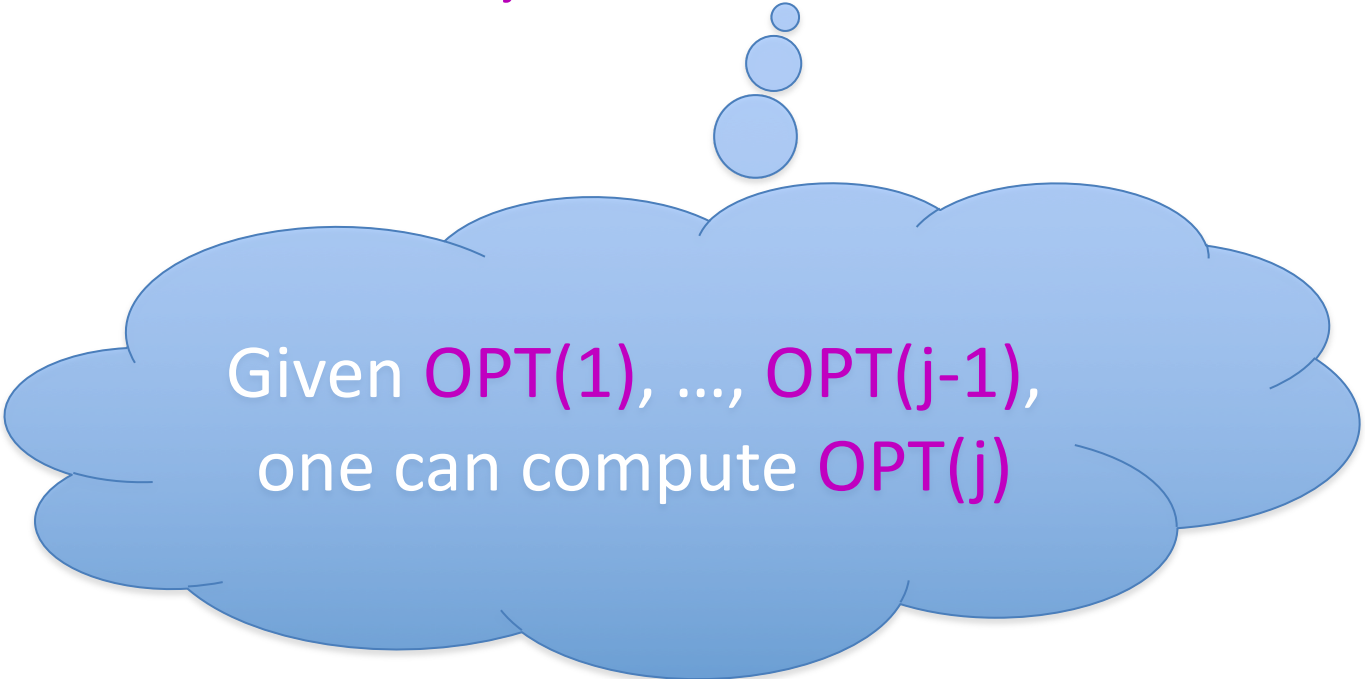
Whenever a recursive call is made an M value is assigned

At most n values of M can be assigned



Property of OPT

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



Given $\text{OPT}(1), \dots, \text{OPT}(j-1)$,
one can compute $\text{OPT}(j)$

Recursion+ memory = Iteration

Iteratively compute the OPT(j) values

Iterative-Compute-Opt

$M[0] = 0$

For $j=1, \dots, n$

$M[j] = \max \{ v_j + M[p(j)], M[j-1] \}$

$M[j] = \text{OPT}(j)$

$O(n)$ run time



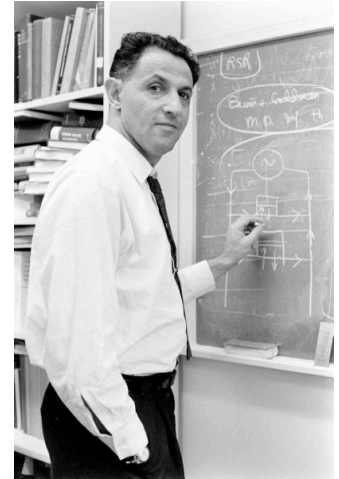
Reading Assignment

Sec 6.1, 6.2 of [KT]



When to use Dynamic Programming

There are polynomially many sub-problems



Richard Bellman

Optimal solution can be computed from solutions to sub-problems

There is an ordering among sub-problem that allows for iterative solution