Lecture 30

CSE 331

Apr 14, 2021

Video mini project

- Due TODAY 8:00pm

- Work with your teammates
- You need to submit one PDF file to Autolab.
- The only thing the PDF needs to have is the link to your video.
- Each group member must submit the exact same PDF

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 $\Sigma_{i \text{ in } S}\, v_j$

Assume: jobs are sorted by their finish time

Couple more definitions

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

= 0 if no such i exists

p(j) < j

OPT(j) = optimal value on instance 1,..,j

Property of OPT

j in OPT(j)

j not in OPT(j)

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

Given OPT(1),…., OPT(j-1), how can one figure out if j in optimal solution or not?

A recursive algorithm

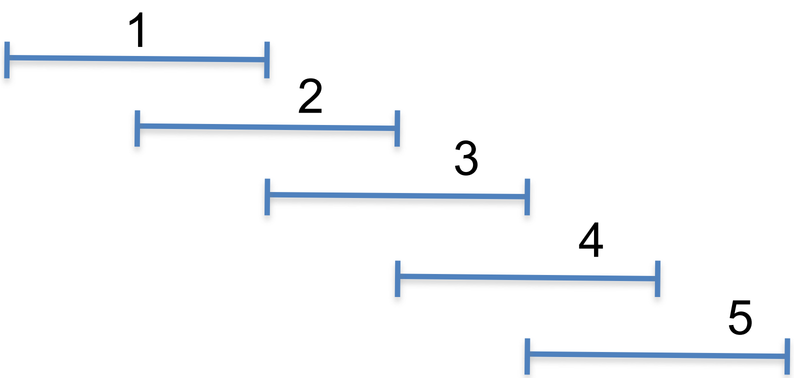
Compute-Opt(j)

```
If j = 0 then return 0

return max { v_j + Compute-Opt( p(j) ), Compute-Opt( j-1 ) }
```

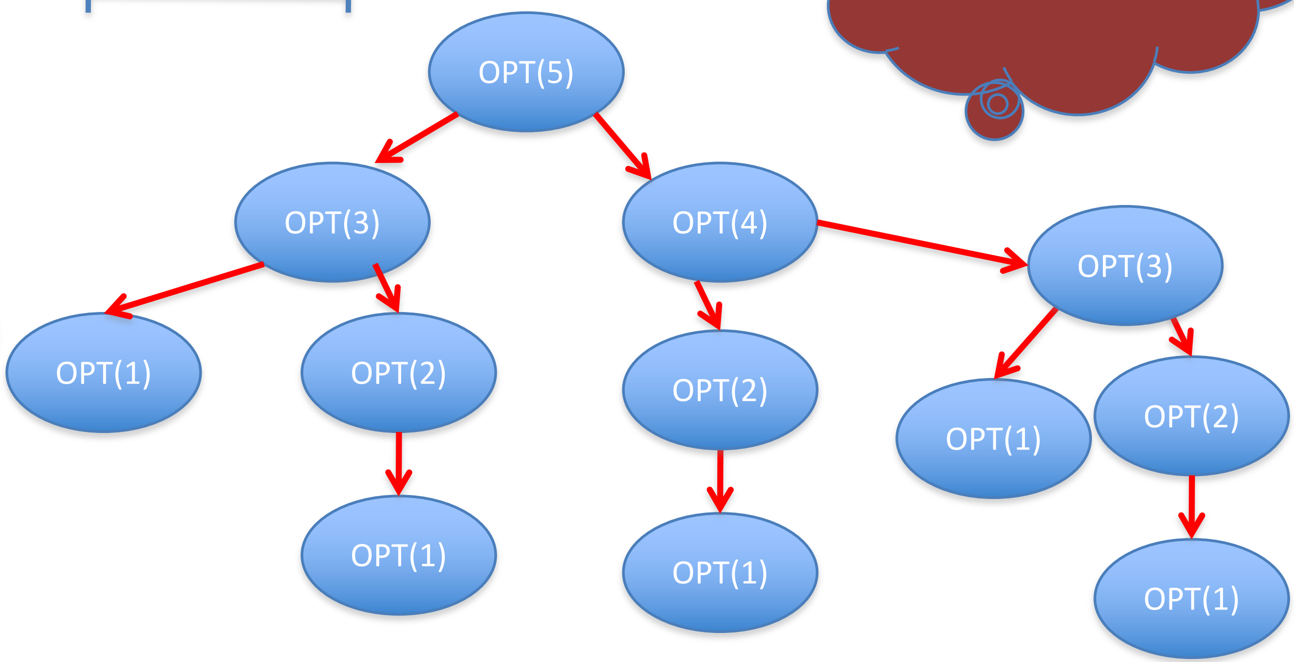
$$OPT(j) = \max \{ v_j + OPT(p(j)), OPT(j-1) \}$$

Exponential Running Time



p(j) = j-2

Only 5 OPT values!



Formal proof: Ex.

A recursive algorithm

M-Compute-Opt(j)

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

If j = 0 then return 0

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

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

return M[j]

Run time = O(# 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-Compute-Opt(p(j)), M-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

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

Given OPT(1), ..., OPT(j-1), one can compute OPT(j)

Recursion+ memory = Iteration

Iteratively compute the OPT(j) values

Iterative-Compute-Opt

M[0] = 0

For j=1,…,n

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

M[j] = OPT(j)

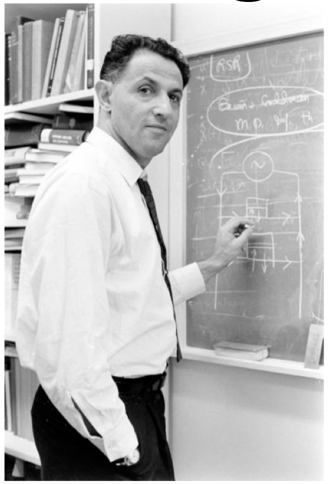
O(n) run time


ICANHASCHEEZBURGER.COM

Reading Assignment

Sec 6.1, 6.2 of [KT]

When to use Dynamic Programming



Richard Bellman

There are polynomially many sub-problems

OPT(1), …, OPT(n)

Optimal solution can be computed from solutions to sub-problems

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

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

OPT (j) only depends on OPT(j-1), …, OPT(1)

Scheduling to min idle cycles

n jobs, i^{th} job takes w_i cycles

You have W cycles on the cloud

What is the maximum number of cycles you can schedule?

Rest of today's agenda

Dynamic Program for Subset Sum problem