# Lecture 33

CSE 331

Nov 18, 2016

# Mini project video grading

# Homework 9

# Homework 9

Due by **12:30pm, Friday, December 2, 2016**.

Make sure you follow all the homework policies.

All submissions should be done via Autolab.

## Question 1 (Programming Assignment) [40 points]

</> **Note**

This assignment can be solved in either Java, Python or C++ (you should pick the language you are most comfortable with). Please make sure to look at the supporting documentation and files for the language of your choosing.

**! Note on Timeouts**

**For this problem the total timeout for Autolab is 480s, which is higher the the usual timeouts of 180s or 240s in the earlier homeworks.** So if your code takes a long time to run it'll take longer for you to get feedback on Autolab. **Please start early to avoid getting deadlocked out before the feedback deadline.**

Also for this problem, `C++` and `Java` are way faster. The 480s timeout was chosen to accommodate the fact that Python is much slower than these two languages.

# HW 8 solutions

End of the lecture

# Graded HW 6

Done by today

Apologies for the delay!

# CS Ed week (Dec 5)

# 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

Proof of correctness by induction on $j$

Compute-Opt($j$)

Correct for $j=0$

If $j = 0$ then return $0$

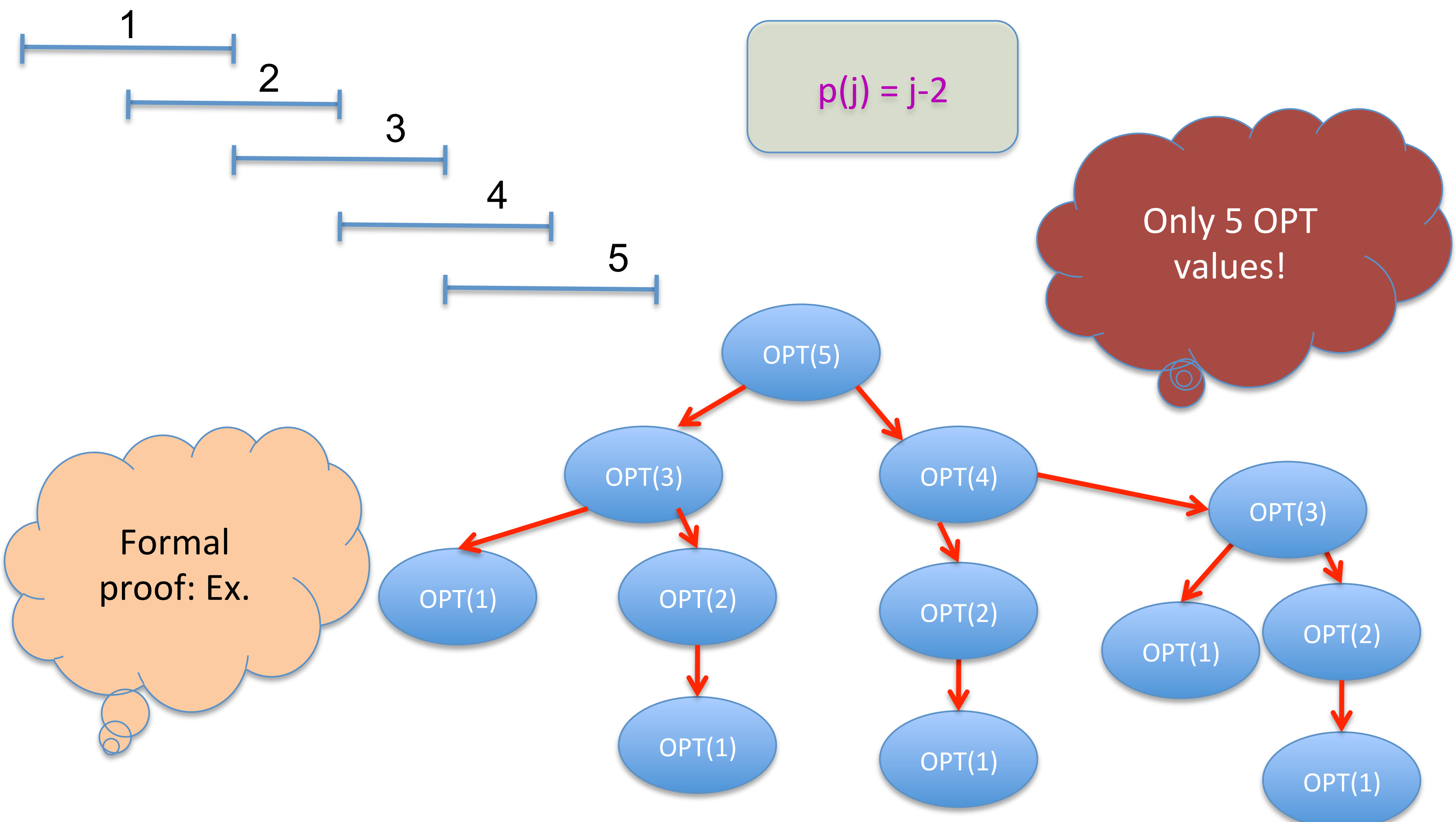
return max { $v_j$ + Compute-Opt( $p(j)$ ), Compute-Opt( $j-1$ ) }

$= OPT( p(j) )$

$= OPT( j-1 )$

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

# Exponential Running Time



# Using Memory to be smarter

```
Pow (a,n)
        .
        .
        .
        .
    // n is even and ≥ 2
    return Pow(a,n/2) * Pow(a, n/2)
        .
        .
        .
        .
```

O(n) as we recompute!

```
Pow (a,n)
        .
        .
        .
        .
    // n is even and ≥ 2
    t= Pow(a,n/2)
    return t * t
        .
        .
        .
        .
```

O(log n) as we compute only once

# How many distinct OPT values?

# 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 $j=1,\dots,n$

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

$M[j] = OPT(j)$

$O(n)$ run time

# Reading Assignment

Sec 6.1, 6.2 of [KT]

# When to use Dynamic Programming

Richard Bellman

There are polynomially many sub-problems

Optimal solution can be computed from solutions to sub-problems

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