Lecture 32

CSE 331

Nov 16, 2016

Talk today

note | stop following | 47 views | Actions

Buffalo Talks | How to crack the coding interviews?

Buffalo Talks has their next talk ready! Sai Vikneshwar, a theoretical computer science major at UB is going to talk about "How to crack the coding interviews?".
He has interviewed with tonnes of companies and failed many times, but there were times when he didn't fail and got offers from companies like Google, Microsoft, Fb, etc. Sai is going to share all his experiences and how he did it.
I assure you, Sai's talk will serve as a great motivation for many of you who are currently in the interviewing process, so don't miss out on this one.
Date and Time - 11/16 (Wed) at 6:00 pm
Venue - Knox 109

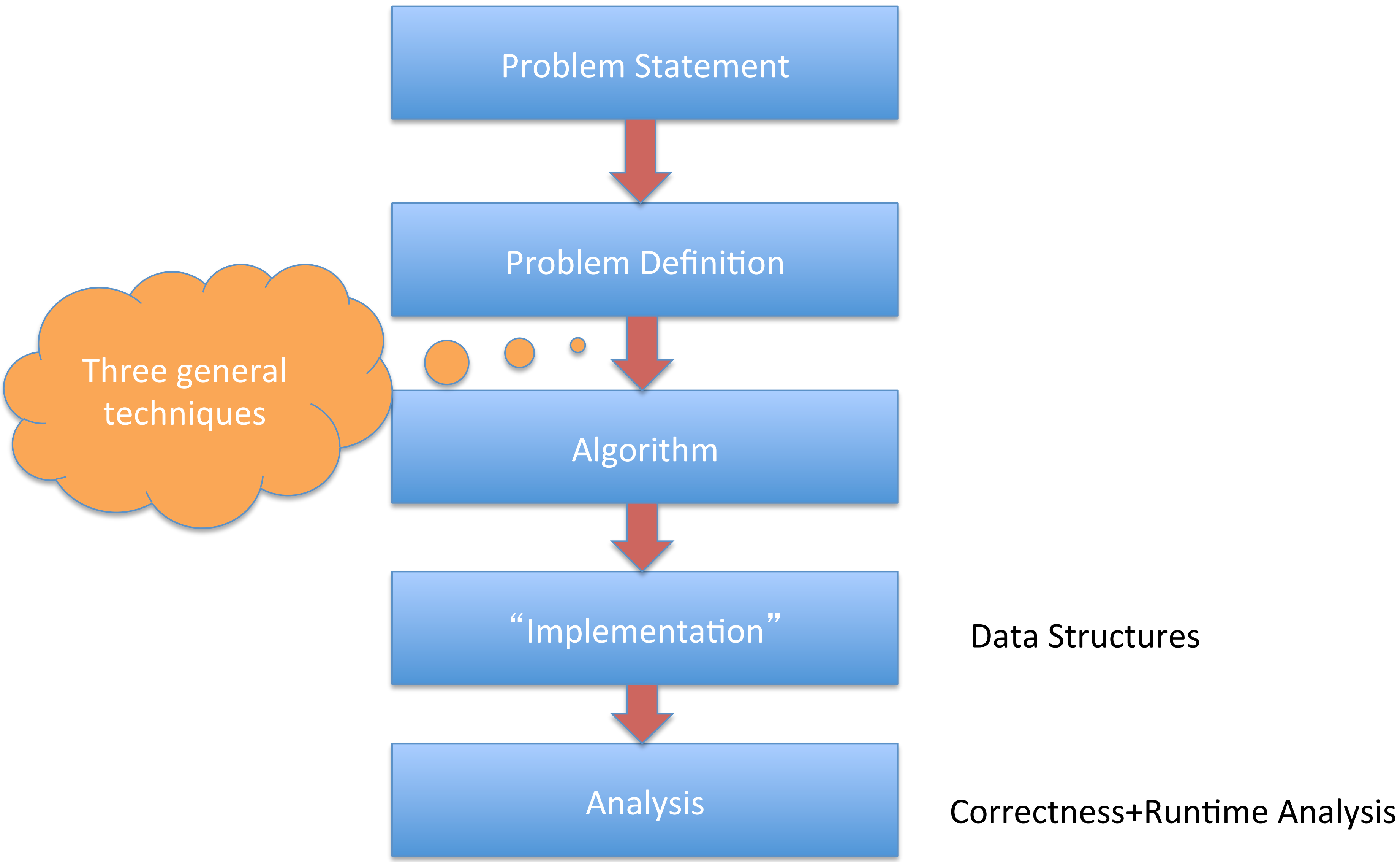
society

~ An instructor (Atri Rudra) thinks this is a good note ~

edit | undo good note | 2

Updated 9 hours ago by Tim Wegner

High level view of CSE 331



Greedy Algorithms

Natural algorithms

Reduced exponential running time to polynomial

Divide and Conquer

Recursive algorithmic paradigm

Reduced large polynomial time to smaller polynomial time

A new algorithmic technique

Dynamic Programming

Dynamic programming vs. Divide & Conquer

Same same because

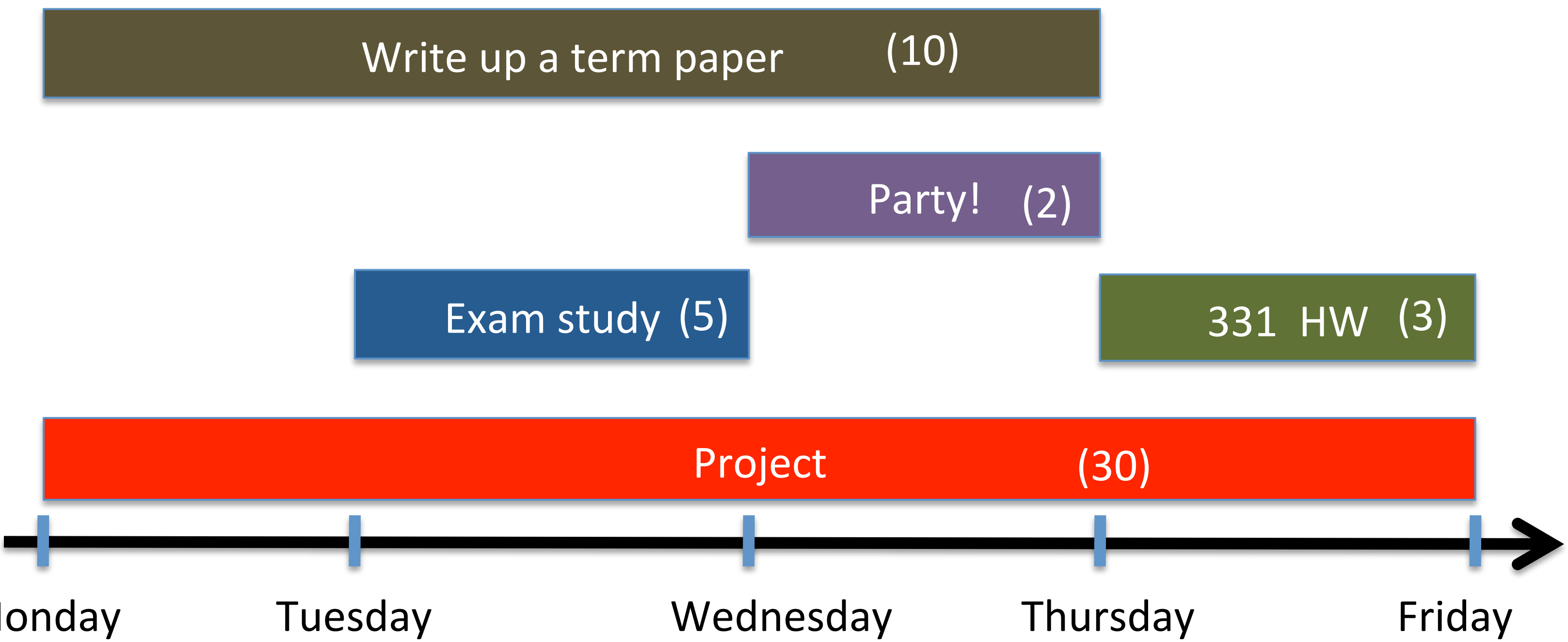
Both design recursive algorithms

Different because

Dynamic programming is smarter about solving recursive sub-problems

End of Semester blues

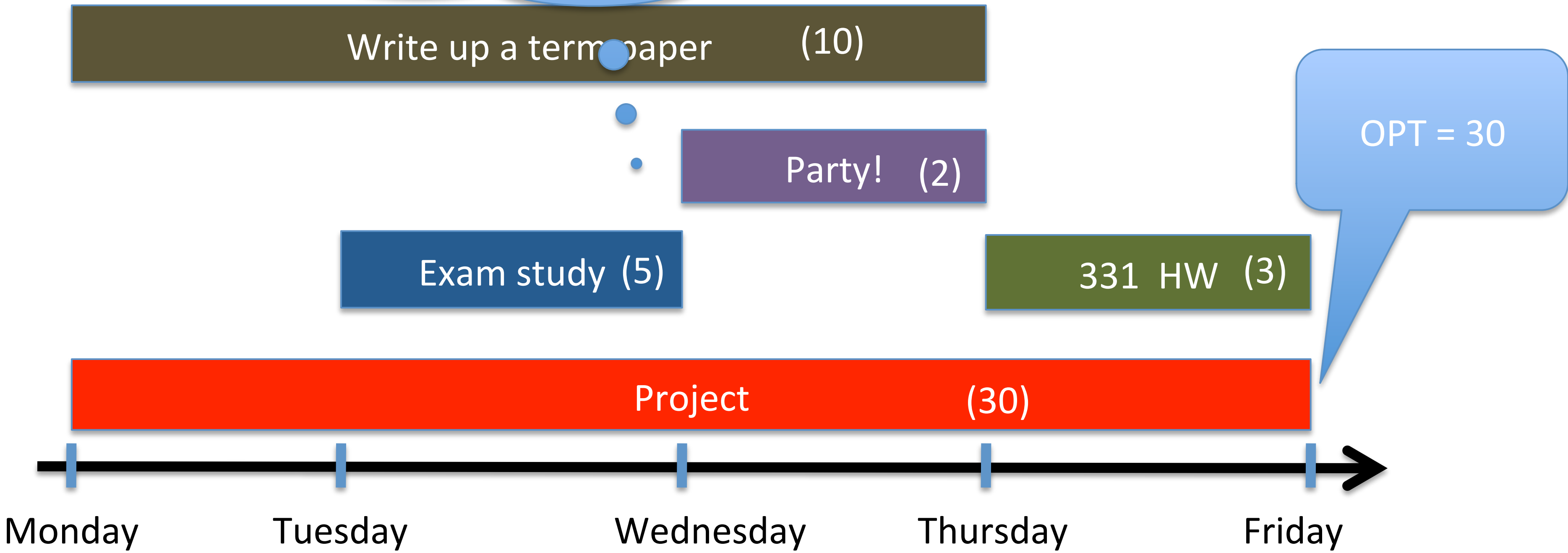
Can only do one thing at any day: what is the optimal schedule to obtain maximum value?



Previous Greedy algorithm

Order by end time and pick jobs greedily

Greedy value = 5+2+3= 10



Today’s agenda

Formal definition of the problem

Start designing a recursive algorithm for the problem

Property of OPT

j in OPT(j)

j not in OPT(j)

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

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

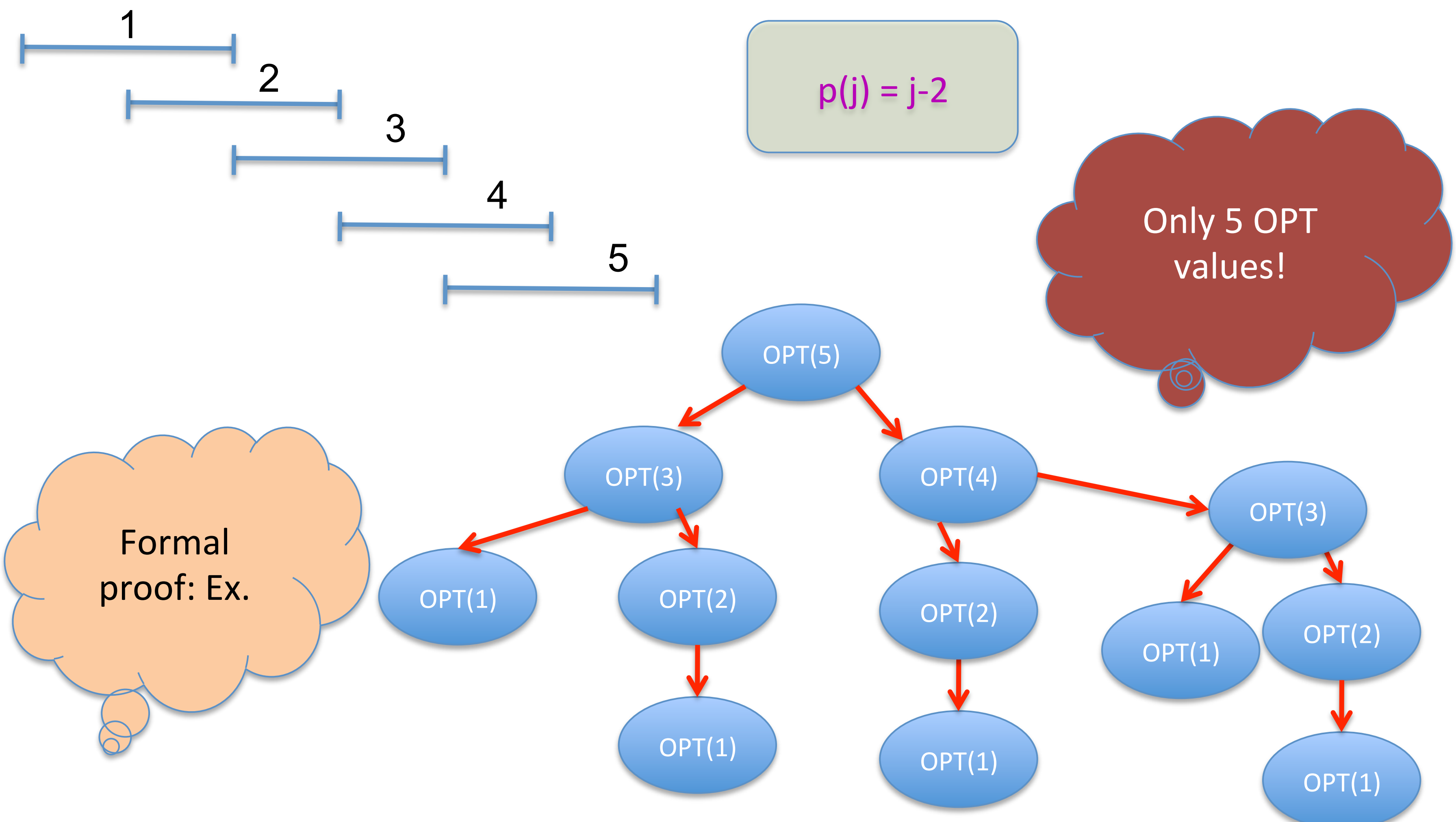
return max { v_j + Compute-Opt(p(j)), Compute-Opt(j-1) }

= OPT(p(j))

= OPT(j-1)

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

Exponential Running Time



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 + M-Compute-Opt(p(j)), M-Compute-Opt(j-1) }

return M[j]

M-Compute-Opt(j) = OPT(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

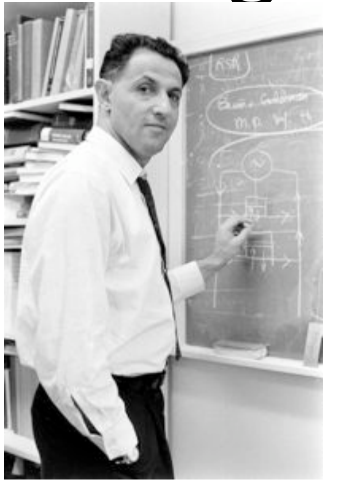
Whenever a recursive call is made an M value of assigned

At most n values of M can be assigned

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