NPTEL MOOC,JAN-FEB 2015
Week 7, Module 1

DESIGN AND ANALYSIS OF ALGORITHMS

Dynamic Programming

MADHAVAN MUKUND, CHENNAI MATHEMATICAL INSTITUTE
http://www.cmi.ac.in/~madhavan

Inductive definitions

- Factorial
  - $f(0) = 1$
  - $f(n) = n \times f(n-1)$
- Insertion sort
  - isort([ ]) = [ ]
  - isort($[x_1,x_2,..,x_n]$) = insert(x_1,isort($[x_2,\ldots,x_n]$))

... Recursive programs

```
int factorial(n):
  if (n <= 0)
    return(1)
  else
    return(n*factorial(n-1))
```

Optimal substructure property

- Solution to original problem can be derived by combining solutions to subproblems
- factorial(n-1) is a subproblem of factorial(n)
  - So are factorial(n-2), factorial(n-3), …, factorial(0)
- isort([x_2,…,x_n]) is a subproblem of isort([x_1,x_2,…,x_n])
  - So is isort([x_i,…,x_j]) for any $1 \leq i \leq j \leq n$

Interval scheduling

- CMI has a special video classroom for delivering online lectures
- Different teachers want to book the classroom — the slot for each instructor i starts at s(i) and finishes at f(i)
- Slots may overlap, so not all bookings can be honoured
- Choose a subset of bookings to maximize the number of teachers who get to use the room

Subproblems

- Each subset of booking requests is a subproblem
- Greedy strategy
  - Pick one request among those still in contention
  - Eliminate bookings that conflict with this choice
  - Solve the resulting subproblem

Subproblems …

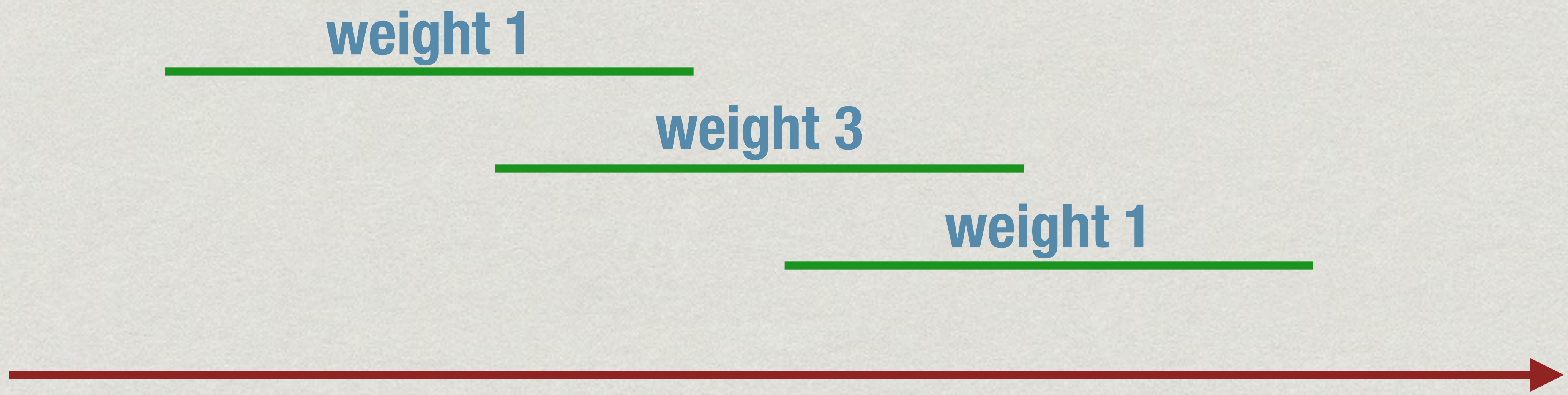
- Each subset of booking requests is a subproblem
- Given N bookings, we have 2^N subproblems
- Greedy strategy efficiently looks at only O(N) of these subproblems
    - Each local choice rules out large number of subproblems
    - Need a proof that this is a valid strategy

Weighted interval scheduling

- Same scenario as before, but each request comes with a weight
  - Weight could be the amount a person is willing to pay for using the resource
- Aim is now to maximize the total weight of the bookings selected
  - Not the same as maximizing the number of bookings selected

Weighted interval scheduling

- Greedy strategy for unweighted case
  - Select request with earliest finish time
- Not valid any more



Weighted interval scheduling

- We can search for another greedy strategy that works …
- … or look for an inductive solution that is “obviously” correct

Weighted interval scheduling

- Let the bookings be ordered by starting time
- Begin with b_1
    - Either b_1 is in the optimal solution or it is not
    - If we include b_1, eliminate conflicting requests from $b_2,\ldots,b_N$ and solve the resulting subproblem
    - If we exclude b_1, solve the subproblem $b_2,\ldots,b_N$
    - Evaluate both options, choose the maximum

Weighted interval scheduling

- The inductive solution considers all options
  - For each b_j, the best solution either has b_j or does not
  - For b_1, we are explicitly checking both cases
  - If b_2 is not in conflict with b_1, it will be considered in both subproblems after choosing b_1
  - If b_2 is in conflict with b_1, it will be considered in the subproblem where b_1 is not chosen
  - …

The challenge

- b_1 and b_2 in conflict, but both compatible with $b_3,b_4,\ldots,b_N$
  - Choose $b_1 \Rightarrow$ subproblem $b_3,b_4,\ldots,b_N$
  - Discard $b_1 \Rightarrow$ subproblem $b_2,b_3,\ldots,b_N$
    - Next stage, choose/discard b_2
    - Discard $b_2 \Rightarrow$ again subproblem $b_3,b_4,\ldots,b_N$

The challenge …

- Inductive solution can give rise to same subproblem at different stages
- Naive recursive implementation will evaluate each instance of same subproblem from scratch
- How do we avoid this wasteful recomputation?
- Memoization and dynamic programming