EECS 336: Introduction to Algorithms **Lecture 9**
Dynamic Programming Weighted Interval Scheduling

Announcements:

- midterm next tuesday, in class.

Reading: 6.0-6.3

Last time:

- Polynomial Multiplication
- Fast Fourier Transform

Today:

- Dynamic Programming
- Weighted interval scheduling

Dynamic Programming

"divide problem into small number of subproblems and **memoize** solution to avoid redundant computation"

Example: Weighted Interval Scheduling

input:

- n jobs $J = \{1, \ldots, n\}$
- s_i = start time of job i
- f_i = finish time of job i
- v_i = value of job i

output: Schedule $S \subseteq J$ of compatible jobs with maximum total value.

Recall Greedy: "earliest finish time"

```
      1
|---------|
                   100
     |-------------|
```

Idea: job i is either in OPT(J) or not.

1. let J' = jobs compatible with i in J.
2. let V = value of OPT if "$i \in$ OPT(j)".

$= v_i + \mathrm{OPT}(J')$

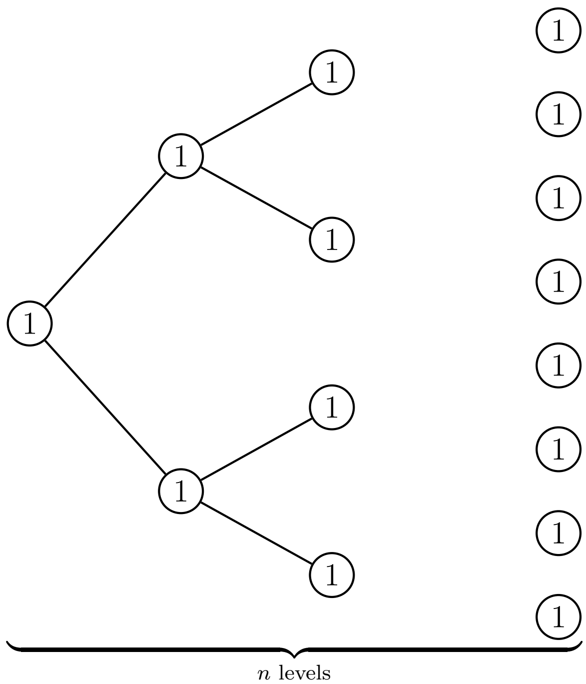
3. let V' = vale of OPT if "$i \notin \mathrm{OPT}(j)$"

   $= \mathrm{OPT}(J \setminus \{i\})$.

4. return $\mathrm{OPT}(J) = \max(V, V')$.

Note: subproblems: schedule J' and $J \setminus \{i\}$.

Recurrence: $T(n) = 2T(n-1) + 1$



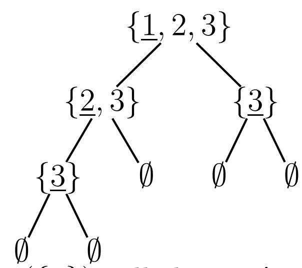
$T(n) = O(2^n)$

Challenge 1: redundant computation

Example:

```
  1      3
|---|  |---|

|---------|
     2
```



Note: OPT({3}) called twice!

Solution: memoize

"when computing the value of a subproblem save the answer to avoid computing it again"

Result: runtime = # of subproblems × cost to combine.

Challenge 2: could have too many subproblems.
(could be exponential!)

Solution: require "succinct description" of subproblems.

Idea: for interval scheduling, process jobs in order of start time so subproblems suffixes of order.

- sort jobs by increasing start time, $s_1 \leq s_2 \leq \cdots \leq s_n$.
- let $\text{next}[i]$ denote job with earliest start time after i finishes. (if none, set $\text{next}[i] = n + 1$)
- subproblems when processing job 1:
    - $J' = \{\text{next}[i], \text{next}[i] + 1, \cdots, n\}$
    - $J \setminus \{i\} = \{2, 3, \ldots, n\}$
- suffix {j,...,n} is succinctly described by "j".

Algorithm: Weighted Interval Scheduling:

1. sort jobs by increasing start time.
2. initialize array $\text{next}[i]$.
3. initialize $\text{memo}[i] = \emptyset$ for all i.
4. initialize $\text{memo}[n + 1] = 0$.
5. compute OPT(1).

Subroutine: OPT(i)

1. if $\text{memo}[i] \neq \emptyset$, return $\text{memo}[i]$.
2. $\text{memo}[i] \leftarrow$
   $\max(v_i + \text{OPT}(\text{next}[i]), \text{OPT}(i + 1))$.
3. return memo(i).

Correctness

"OPT(i)" is correct (by induction on i)

Runtime Analysis

- n subproblems
- constant time to combine
- initialization: sorting & precomputing next array

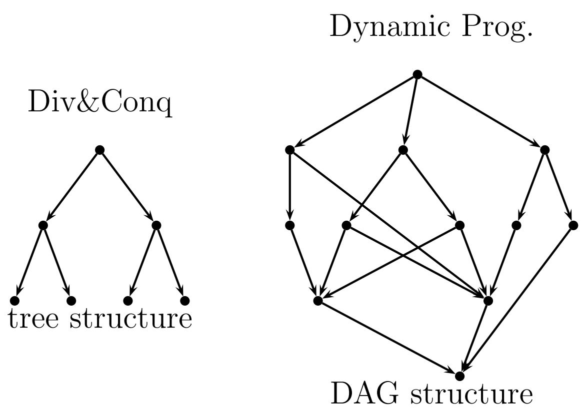
Runtime: $O(n)+$ initialization $= O(n \log n)$

Key Ideas of Dynamic Programming

Subproblems must be:

1. succinct
   (only a polynomial number of them)
2. efficiently combinable.
3. partially ordered (avoid infinite loops), e.g.,
   - process elements "once and for all"
   - "measure of progress/size".

Comparison to Divide and Conquer



Iterative DPs

"fill in memoization table from bottom to top"

Algorithm: iterative weighted interval scheduling

1. memo$[n+1] = 0$.
2. for $i = n$ down to 1.

   memo$[i]$ = $\max(v_i +$ memo[next(i)], memo$[i+1]$).

Finding Optimal Schedule

"traverse memoization table to find schedule"

Algorithm: schedule

$i = 1$

while $i < n$

if memo$[i+1] < v_i +$ memo[next(i)]

schedule i; $i \leftarrow$ next(i).

else

$i \leftarrow i + 1$.

endif

endwhile

Suggested Approach

I. identify subproblem in english

OPT(i) = "optimal schedule of $\{i, \ldots, n\}$ (sorted by increasing start time)"

II. specify sumbroblem recurrence

OPT(i) = $\max($OPT$(i+1), v_i +$ OPT(next(i)))

III. identify base case

OPT$(n+1) = 0$

IV. write iterative DP.