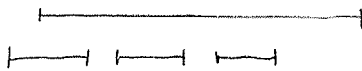


What if we want to maximize total length of the set of nonoverlapping jobs? (job length = $f_i - s_i$) (4/6)

Previous greedy algorithm doesn't work

Example: 

Given $S = (s_1, f_1) (s_2, f_2) \dots (s_n, f_n)$

Find set of non-overlapping jobs of maximum total length.

Observation An optimal solution for jobs S

either

① includes job n and contains an optimal solution for $S' =$ jobs not overlapping job n

or ② doesn't include job n and is an optimal solution for jobs $1 \dots n-1$

In both cases we have created a subproblem of our original problem, which we can solve by creating more subproblems.

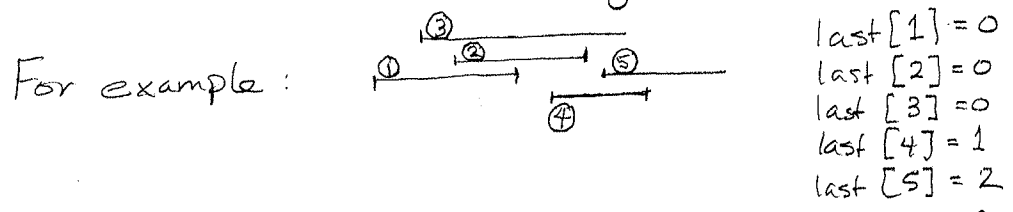
How many subproblems do we need to solve?

Because of ② we need to solve all problems of the form: Find an optimal set from jobs $1 \dots i$

What about ①? Can we order the jobs so that $S' =$ jobs $1 \dots i$ for some i ?

Yes! If $f_1 \leq f_2 \leq f_3 \dots \leq f_n$ then the jobs that don't overlap job n are jobs $1 \dots i$ for some i .

Let $last[j] =$ largest index less than j of a job that doesn't overlap job j (or 0 if no such job exists)



Let $L[j] =$ total length of optimal solution for jobs $1 \dots j$

Observation $L[n] = \max \left\{ \underset{\textcircled{1}}{L[last[n]] + (f_n - s_n)}, \underset{\textcircled{2}}{L[n-1]} \right\}$

In general:
 $L[j] = \max \left\{ L[last[j]] + (f_j - s_j), L[j-1] \right\}$
 for $j = 1, 2, \dots, n$

where $L[0] = 0$

Max Length Sched (S)

- $O(n \lg n)$ (A) Sort jobs so that $f_1 \leq f_2 \leq \dots \leq f_n$ Use binary search to
- $O(n \lg n)$ (B) Calculate $last[j]$ for $j = 1 \dots n$ How? find s_j in sorted list of f_1, f_2, \dots, f_n
- (C) $L[0] = 0$
- $O(n)$ (D) For $j = 1$ to n
 $L[j] = \max \left\{ L[last[j]] + (f_j - s_j), L[j-1] \right\}$
- (E) Return $L[n]$

↖ This returns the value (total length) of the optimal solution.
 What if we want the set of jobs?

- (E) $W = \emptyset$
- (F) $j = n$
- (G) while $j > 0$
 [if $L[j] > L[j-1]$ then $\{ W = W \cup \{j\}; j = last[j] \}$
 else $\{ j = j - 1 \}$
- (H) Return W

This is an example of Dynamic Programming:

To solve a problem

- Optimal substructure {
- Find a relation that shows how to solve the problem given solutions to "smaller" subproblems
 - (make sure the total number of subproblems is small)
 - Solve the subproblems from smaller to larger

Longest Increasing Subsequence

Given a sequence of numbers $A = a_1, a_2, \dots, a_n$

a subsequence of A is a sequence $B = b_1, b_2, \dots, b_k$

such that $b_1 = a_{i_1}, b_2 = a_{i_2}, \dots, b_k = a_{i_k}$

and $1 \leq i_1 < i_2 < \dots < i_k \leq n$

For example: $A = 3 \ 5 \ 2 \ 1 \ 7 \ 10 \ 9$

has subseq. $B = 5 \ 1 \ 9$ ($i_1=2 \ i_2=4 \ i_3=7$)

A sequence $B = b_1, b_2, \dots, b_k$ is increasing if $b_1 < b_2 < \dots < b_k$

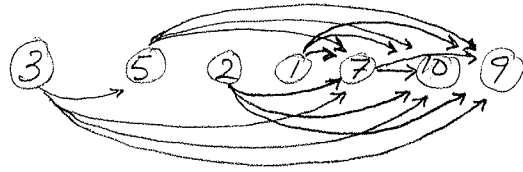
Find a longest increasing subsequence of $A = a_1, a_2, \dots, a_n$.

for $3 \ 5 \ 2 \ 1 \ 7 \ 10 \ 9$ one answer is $3, 5, 7, 9$

How do we find LIS(A)?

Consider all possible increasing subsequences?

No. Too many.



Intuition

Find the longest path in this graph
 where vertex i has label a_i $i=1 \dots n$
 and edge (i,j) exists iff $a_i < a_j$ and $i < j$

Let $L[j]$ = length of the longest increasing subsequence of A that ends with a_j

$$L[j] = 1 + \max \{ L[i] \mid i < j \text{ and } a_i < a_j \}$$

Algorithm (to find length of LIS(A))

$O(n^2)$ [for $j = 1$ to n
 $O(n) \rightarrow L[j] = 1 + \max \{ L[i] \mid i < j \text{ and } a_i < a_j \}$
 $O(n) \rightarrow$ return $\max_j L[j]$

An LIS that ends at a_j is an LIS that ends at a_i (for $i < j$ and $a_i < a_j$) followed by a_j .

To find the actual sequence:

- Let k be such that $L[k] = \max_j L[j]$

print k

for $i = k-1$ to 1

[if $a_i < a_k$ and $L[i] = L[k] - 1$ then
 [print i
 [$k = i$