

# Dynamic Programming I

Recursion:

To solve a problem for one input,  
solve on other inputs  
and combine results.

Benefits: often easy to find

Problem: usually exponential time.

How can we make it faster?

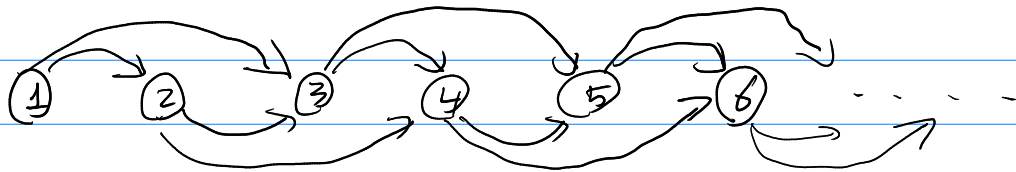
Memoization: whenever you compute  
 $f(x)$  in some recursive call,  
store the answer. Next time  
you call  $f(x)$ , just return it.

Memoization: time = (# possible inputs) · (time per call)

Recursion: time = # paths from input  
to base case

Can think of inputs as a DAG,  
 $A \rightarrow B$  if  $B$ 's recursion uses  $A$ .

Fibonacci:



Recursion time = # paths =  $F_n \approx 1.6^n$   
Memoization time = (# inputs) · (time per input)  
 $= n \cdot n = n^2$  bit operations

Bottom-up DP: fill from left to right,  
looking at each edge.

Can either pull or push along edges.

pull: compute a node's value when you visit it, based on previous values

push: when you visit a computed node, update future nodes that use it.

(In Fibonacci: add your value to theirs)

Generally equivalent. Sometimes only have easy access to out- or in-edges.

# Interval Scheduling:

Want to compute  $Sched(I)$   
where  $I =$  set of  $(s_i, f_i, w_i)$  pairs  
maximize  $\sum_{i \in S} w_i$

for  $S \subseteq I$  non-overlapping.

Naive recursion:

$Sched(I)$

Let  $i =$  first elt of  $I$

Return min of

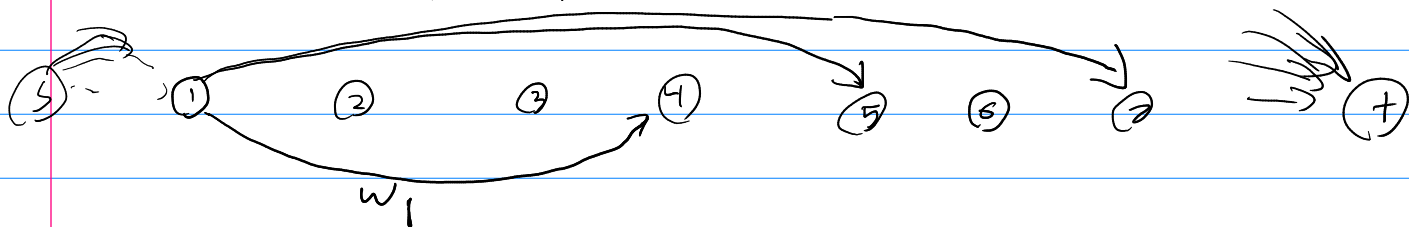
not chosen:  $Sched(I \setminus i)$ ,  
chosen:  $Sched(I \setminus \{i \text{ or anything conflicting with } i\}) + w_i$

Problem:  $2^n$  possible inputs.

Solution: If  $I$  sorted by  $f_i$ , then  
only  $n+1$  inputs ever happen:  
(suffix of  $I$  sorted by  $s_i$ )

$\Rightarrow$  memoized time is  $n \cdot (\text{time per input})$   
 $n^2$  naively  
 $n$  more carefully.  
 $Sched(\text{index in } I, f \text{ of last chosen})$

DAG: sort by  $f_i$ .



$i \rightarrow j$  if  $f_i \leq s_j$  of weight  $w_{ij}$   
 $s \rightarrow$  everything weight 0  
everything  $\rightarrow t$  weight  $w_j$

Answer = max weight  $s \rightarrow t$  Path.

Path  $s \rightarrow i_1 \rightarrow i_2 \rightarrow \dots \rightarrow i_k \rightarrow t$

- weight =  $0 + w_{i_1} + w_{i_2} + \dots + w_{i_k}$   
= total value of scheduling them.

-  $f_{i_1} \leq s_{i_2} < f_{i_2} \leq s_{i_3} < \dots$

$\Rightarrow$  is valid schedule.

Many DP problems have this form:

Convert to a DAG  
Find Max-weight (or min-weight)  
 $s \rightarrow$  path.

Time = # edges in DAG.

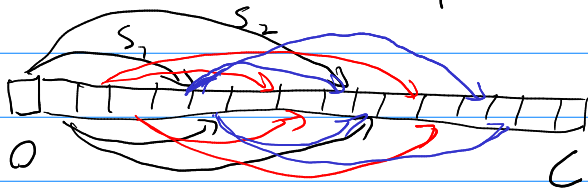
# Stamps

values  $s_1, \dots, s_n$   
want collection of value  $C$ .  
w/ fewest stamps

What is the DAG?

nodes = value

$x \rightarrow x + s_i \quad \forall i, x$ , of weight 1



Answer = shortest  $0 \rightarrow C$  path.

Time = # edges =  $C \cdot N$ .