Dynamic Programming

Solving a problem by decomposing it into smaller subproblems and solving them from smallest to largest

Introduced by Richard Bellman (of Bellman-Ford)

"Programming": scheduling/planning

"Dynamic": because it sounds cool

Fibonacci

0, 1, 1, 2, 3, 5, 8, ...

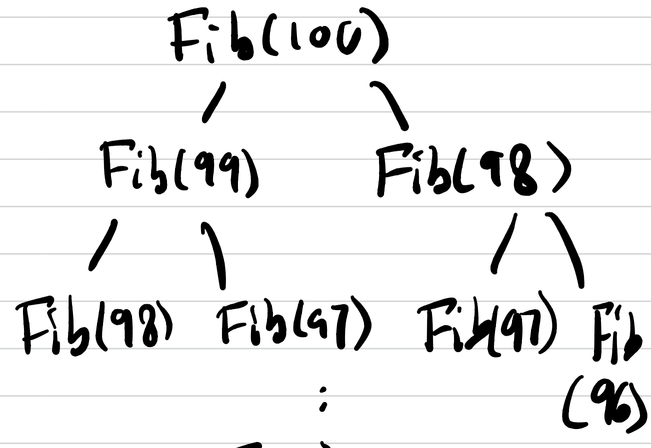
$F(0) = 0, \quad F(1) = 1, \quad F(n) = F(n-1) + F(n-2)$

Fib(n)

if n=0 return 0

if n=1 return 1

return Fib(n-1)+Fib(n-2)

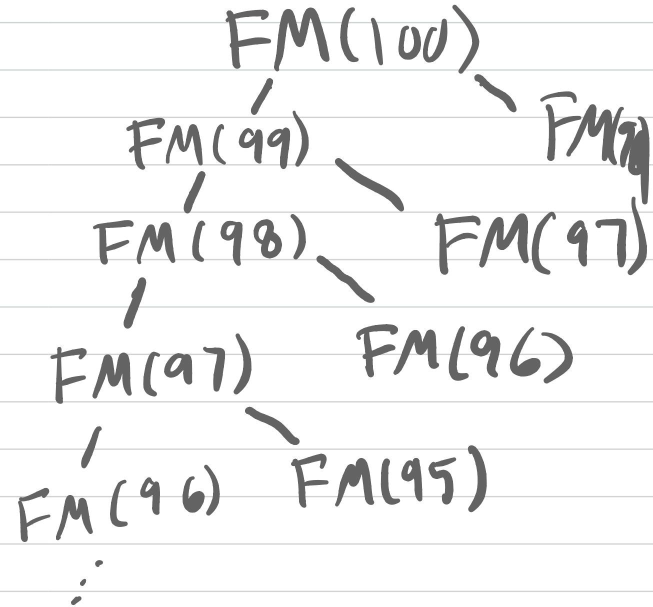


Lecture 1: Fib(n) takes $\geq 1.5^n$ calls to Fib()

Fix: Store values of Fib(n) once computed so we don't have to recompute "memoization"

Fibonacci with memoization

```
FibMemo(n)
    if n = 0 return 0
    if n = 1 return 1
    if n in Memo
        return Memo[n]
    Memo[n] = FibMemo(n-1)
              + FibMemo(n-2)
    return Memo[n]
```



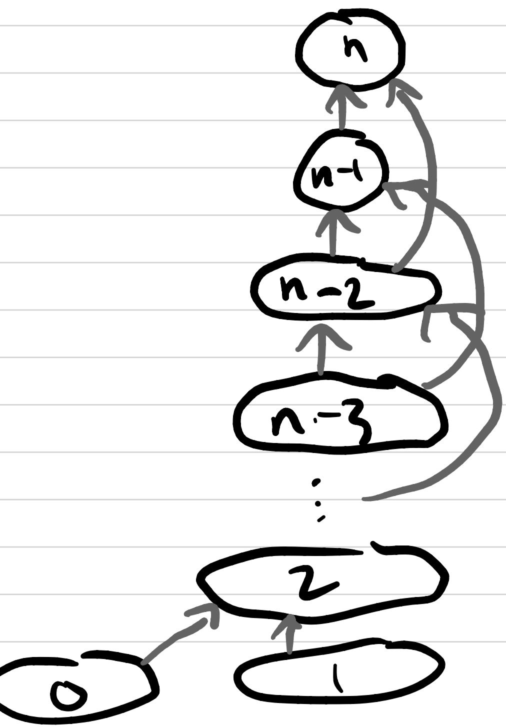
# of recursive calls = $O(n)$

Subproblems: computing $F(0), \ldots, F(n)$

"top-down dynamic programming"

The underlying DAG

View each subproblem as a node
Add a directed edge from $i \to j$
if subproblem j directly depends
on solution to subproblem i



This is a DAG

Should do topological sort, then solve subproblems in that order

FibBottomUp(n)

```
Mem[0] = 0
Mem[1] = 1
for i from 2 to n
    Mem[i] = Mem[i-1] + Mem[i-2]
Return Mem[n]
```

this is "bottom-up dynamic programming"

Outline for designing DP algorithms

1. Identify subproblems ($F(0), \ldots, F(n)$)

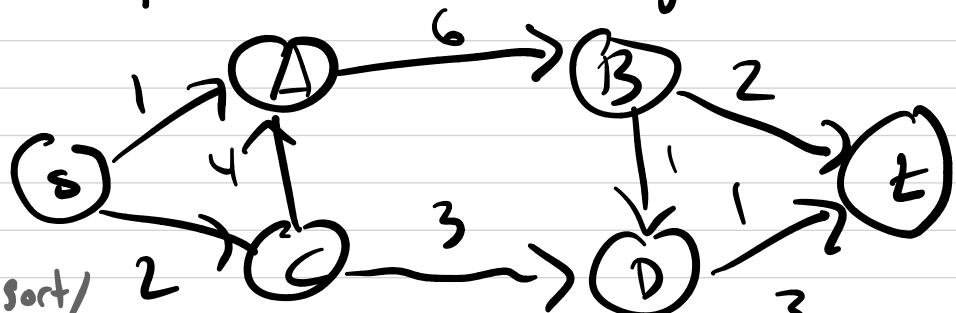
2. Compute recurrence ($F(n) = F(n-1) + F(n-2)$)

3. Design algorithm (bottom-up Fib)

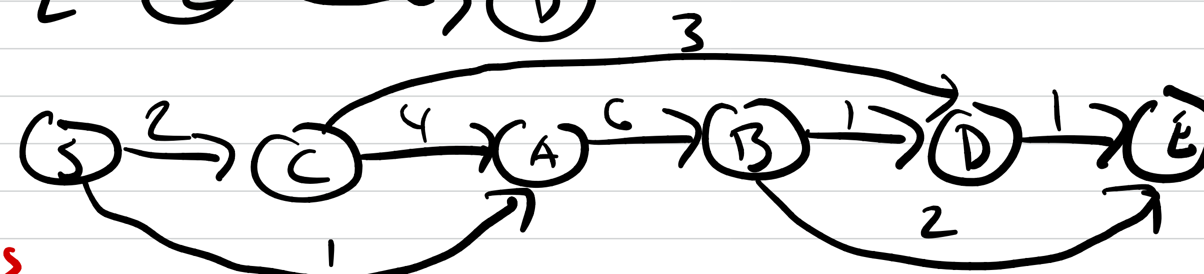
Shortest paths in DAGs

Input: DAG $G=(V,E)$, edge lengths $\ell(u,v)$ (positive or negative)
$s,t \in V$

Output: Compute $dist(t)$ = length of shortest $s \to t$ path



topological sort/ linearize



DP design steps

1. Identify subproblems: $\forall v \in V$, compute $dist[v]$
2. Compute recurrence: $dist(D) = \min\{dist(C)+3,\ dist(B)+1\}$

Recurrence: $\text{dist}[v] = \min_{u:(u,v)\in E} \{\text{dist}[u] + \ell(u,v)\}$

3. Design algorithm

initialize all dist() values to ∞

$\text{dist}(s) = 0$

for each $v \in V \setminus \{s\}$ in linearized order

$\text{dist}[v] = \min_{(u,v)\in E} \{\text{dist}[u] + \ell(u,v)\}$ (*)

return dist[t]

Runtime: $O(n+m)$

Underlying DAG: G itself

What if we had wanted to compute shortest path?

introduce prev[v] = the u that minimizes (*)

Longest increasing subsequences

Input: Sequence of n integers $a_1, \dots, a_n$

Output: Length of longest increasing subsequences (non-contiguous)

$$a = 5 \quad 2 \quad 8 \quad 6 \quad 3 \quad 6 \quad 9 \quad 7$$

DP design steps

1. Identify subproblems: for each prefix $a_1, \dots, a_j$

   Compute: $L[j]$ = length of LIS in $a_1, \dots, a_j$

   ends in a_j

2. Compute recurrence: $L[j] = \max\limits_{i<j} \{ L[i] : a_i < a_j \} + 1$

   if no $a_i < a_j$, $L[j] = 1$

3. Design algorithm

for $j = 1, \ldots, n$

if $\exists\ i<j$ s.t. $a_i < a_j$

$L[j] = 1 + \max\limits_{i<j} \{ L[i] : a_i < a_j \}$

else

$L[j] = 1$

return max $L[j]$ over all j

Runtime: $O(n)$ subproblems $\times$ $O(n)$ time/subproblem

$= O(n^2)$ time

Edit distance

Input: two strings $S[1 \ldots m]$, $T[1, \ldots n]$

Goal: Find smallest number of edits to turn S into T

Edits allowed:
1. insert character into S
2. delete character from S
3. substitute one character for another

Example: S = "S n o w y"
T = "S u n n y"

S n o w y
S $\boxed{u}$ n o w y insert u
S u n n w y replace o → n
S u n n ~~w~~ y delete w

Alignment

S = "Snowy"

T = "Sunny"

S n o w y

S $\boxed{u}$ n o w y

S u n $\boxed{n}$ w y

S u n n ~~w~~ y

"Alignment"

	insert u		replaced o → n	delete w	
S	—	n	o	w	y
S	u	n	n	—	y

Cost: # of columns where symbols differ

Edit distance = minimum cost of an alignment

DP design steps

1. Identify subproblems: for all $0 \le i \le m$, $0 \le j \le n$

$$E(i,j) = \text{EditDistance}(S[1,\dots,i], T[1,\dots,j])$$

$= $ cost of best alignment between $S[1,\dots,i]$, $T[1,\dots,j]$

2. Compute recurrence

$S[1,\dots,i]$ $\quad$ $T[1,\dots,j]$

look at last column in optimal alignment

$$\begin{array}{ccc} S[i] & - & S[i] \\ - & T[j] & T[j] \end{array}$$

$$E(i,j) = \min \begin{cases} 1 + E(i-1,j), \\ 1 + E(i,j-1), \\ E(i-1,j-1) + \text{diff}(i,j) \end{cases}$$

$$\text{diff}(i,j) = \begin{cases} 0 & \text{if } S[i] = T[j] \\ 1 & \text{o.w.} \end{cases}$$

$E(i,0) = i$, $E(0,j) = j$

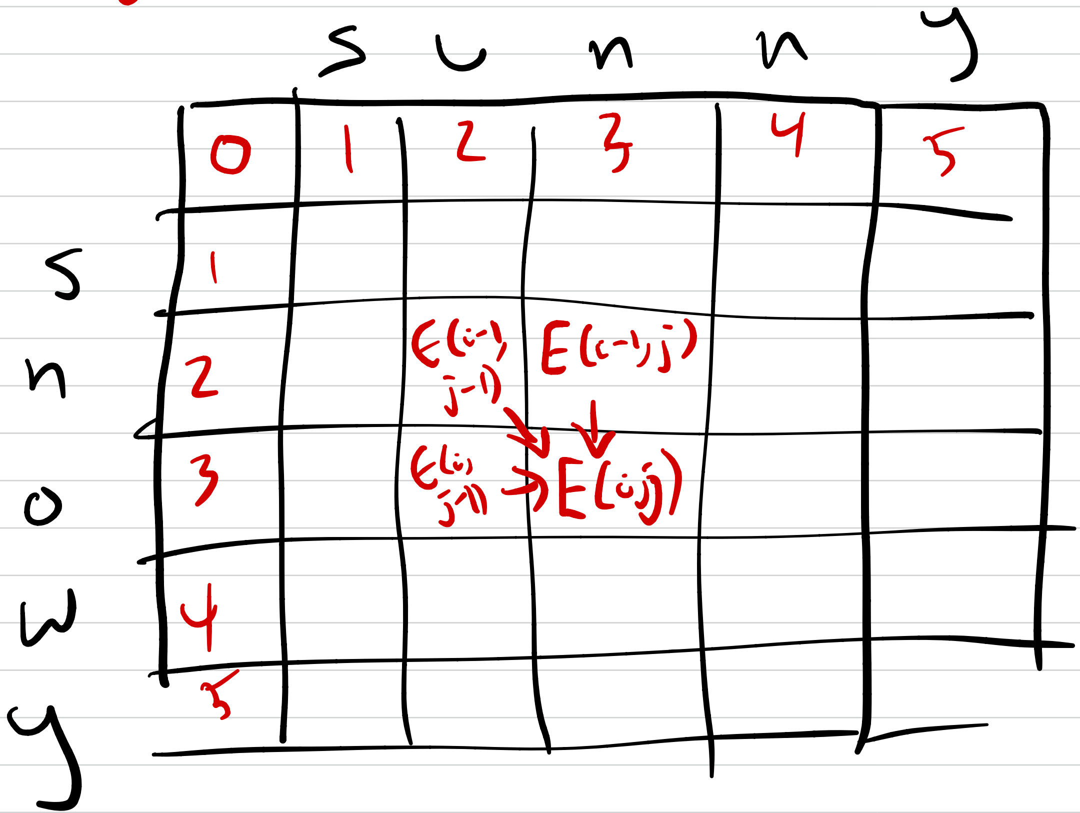
a b

a

b

$$\begin{matrix} a & - \\ - & b \end{matrix}$$

3 Design algorithms



for $i = 0, 1, 2, \ldots, m$

$E(i,0) = i$

for $j = 0, 1, 2, \ldots, n$

$E(0,j) = j$

for $i = 1, 2, \ldots, m$

for $j = 1, 2, \ldots, n$

$E[i,j] = \min\{E(i-1,j)+1,\ E[i,j-1]+1,$
$E[i-1,j-1] + \mathrm{diff}(i,j)\}$

return $E(m,n)$

Runtime:

$O(mn)$

$S = a$

$T = a$

$T' = b$

$E(0,0) = 0$

$E(1,1) = E(0,0) + \mathrm{diff}(1,1)$