CS412 - DYNAMIC PROGRAMMING

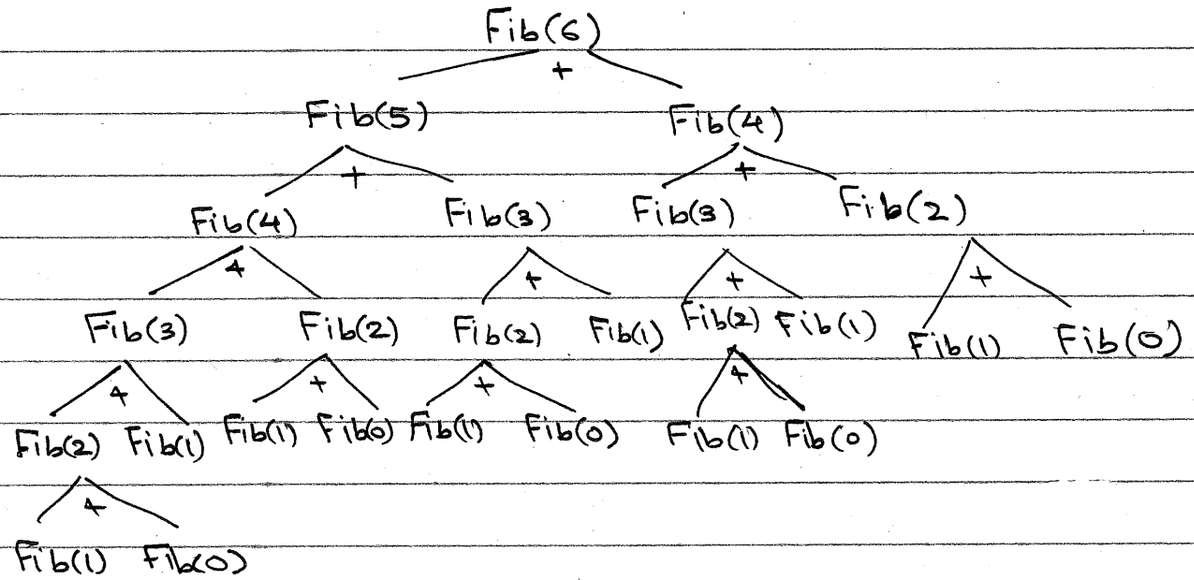
Date

"Dynamic Programming", according to Richard Bellman uses a phrase that not even a congressman could object."

Consider the good, old Fibonacci recurrence:

$$F_n = F_{n-1} + F_{n-2}$$

let's draw the recurrence tree for F_6 (or Fib(6)):



Observe: a) Overlapping Subproblems (same subproblems appearing on both sides of the recurrence tree)

b) An optimal substructure

The two signatures for problems suitable for dynamic programming

There are two key observations:

a) Overlapping Subproblems: Solving many subproblems many times; contrast with Mergesort: A subarray is only sorted once. In Fib(n), we are solving many subproblems many times [really gets bad for large n].

Signature

No.

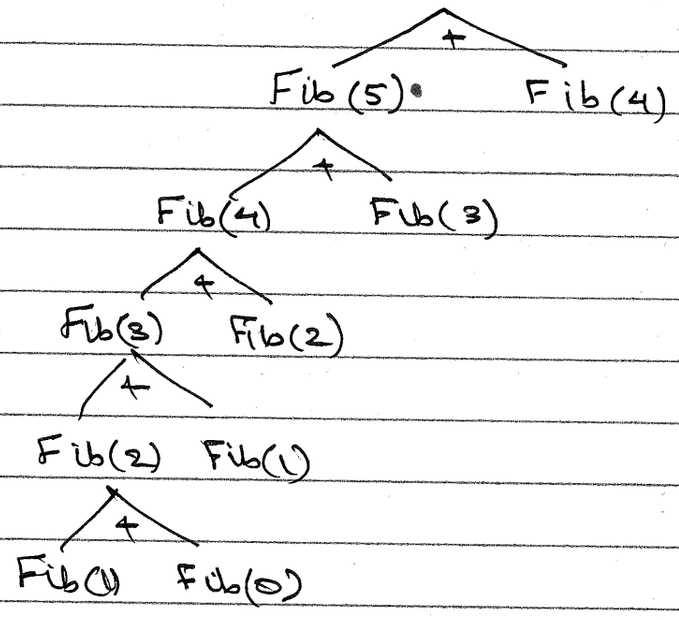
b) Optimal Substructure: If a globally optimal solution can be found by combining [many], optimal solutions to local subproblems
Mergesort does that as a classical example of a divide-and-conquer algorithm.

So, how can we efficiently compute Fib(n), which when solved recursively takes exponential time?

There are two approaches in DP:
a) Use an iterative approach [bottom-up]
b) Still use a recursive approach but keep a memo [memoization] of solutions of already computed local subproblems,

Solving Fib(n) using memoization

Fib(6) {Left-first-depth-first}



Memo [recall, dictionary in Python]

Fib(0) = 1
Fib(1) = 1
Fib(2) = 2
Fib(3) = 3
Fib(4) = 5
Fib(5) = 8
Fib(6) = 13

Signature _____ No. _____

Date

In the 1950s, Richard Bellman, an applied mathematician employed at RAND Corporation, [famous for the Bellman-Ford algorithm and coining the term 'curse of dimensionality'] was working on the shortest-path problem of dags.

The Cold War had begun. What Bellman was working on, had to be justified as something 'applied'. There he coined the term 'dynamic programming' to hide that he was working on something that may have no application at all!

So, when does DP (aka Tabular method) apply?

1) When we have a recursive approach and representation of an optimization problem.

2) Can break the problem into a combination of subproblems.

3) Use solutions [optimal] to subproblems to solve the original optimization problem.

4) Unlike divide-and-conquer, eg., $T(n) = a\,T(\frac{n}{b}) + f(n)$, subproblems get smaller only by a slight difference eg., $F_n = F_{n-1} + F_{n-2}$, i.e.,

to find F_{100}, we need to compute F_{99} and F_{98}, which are only slightly smaller than the actual problem.

5.) Overlapping subproblems.

Signature

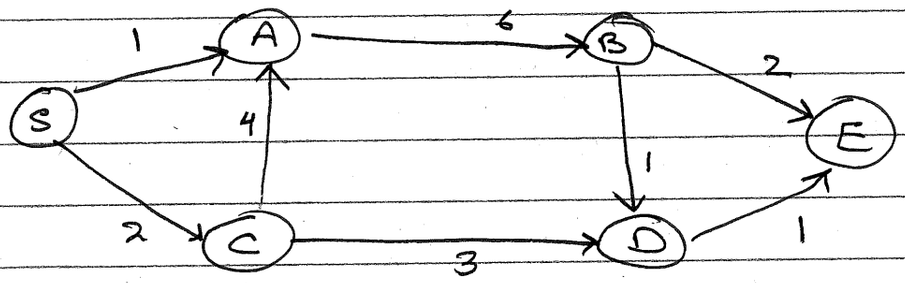
No.

It all began with dags!

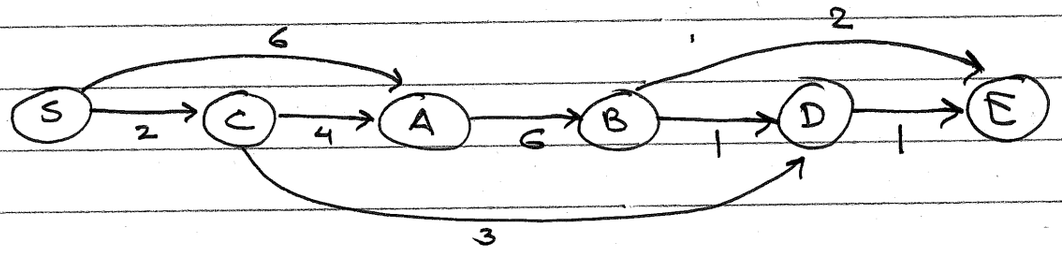
Finding Shortest Path in Dags (the basis of DP)

Given a dag, vertices appear in a topological order (linearized), i.e., in decreasing finishing time.

Given a dag, $G=(V,E)$ with weighted edges,



We run DFS on G, to obtain the linearized graph:



Claim: To find a shortest path to a node from a source node, it is sufficient to linearize the dag.

1) Smallest optimal subproblem: dist(S,S) = 0 or dist(S) = 0 [in short form].

2) Start building the solution to bigger problems using optimal solutions to subproblems.

So,

$$\underbrace{dist(v)}_{\text{from source node 's'}} = \min_{(u,v)\in E} \left\{ \underbrace{dist(v)}_{\text{initialized by } \infty}, \; dist(u) + c(u,v) \right\}$$

a recursive definition!

Let's march on the topological order (visiting each node in the dag in order):

$dist(s) = 0$ [the base-case]

$$dist(c) = \min \left\{ \underbrace{dist(c)}_{\infty}, \; \underbrace{dist(s)}_{=0} + \underbrace{c(s,c)}_{=2} \right\}$$

$$= 2$$

$$dist(A) = \min \left\{ \underbrace{dist(A)}_{\infty}, \; \underbrace{dist(s)}_{=0} + \underbrace{c(s,A)}_{=1}, \; \underbrace{dist(c)}_{=2} + \underbrace{c(c,A)}_{=4} \right\}$$

$$= 1$$

and so on!

Likewise,

$$dist(B) = \min \left\{ \underbrace{dist(A)}_{=1} + \underbrace{c(A,B)}_{=6} \right\} = 7$$

$$dist(D) = \min \left\{ \underbrace{dist(B)}_{=7} + \underbrace{c(B,D)}_{=1}, \; \underbrace{dist(c)}_{=2} + \underbrace{c(c,D)}_{=3} \right\}$$

$$= 5$$

and,

$$dist(E) = \min \left\{ \underbrace{dist(B)}_{=7} + \underbrace{c(B,E)}_{=2}, \; \underbrace{dist(D)}_{=5} + \underbrace{c(D,E)}_{=1} \right\}$$

$$= 6$$

Date

Now, Bellman's idea went beyond this: Imagine that the nodes of a dag are subproblems of some large problem, and finding the shortest path from the source [base-case] to every other node is the original problem

⇒ All dynamic programming problems can be reduced to the shortest path problem in a dag (only, the dag remains implicit).

×——————×

Finding the Longest Common Subsequence Problem (aka LCS)

Let X and Y be two strings or sequences. We are looking for the longest common (if any) subsequence of X and Y.

So, what is a subsequence? A subset of a string / sequence that is in order.

For example, given a sequence of 'n' numbers, say, $a_1, a_2, \ldots, a_n$, a subsequence is any subset of the form $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$

s.t.,

$$1 \leq i_1 < i_2 < \ldots < i_k$$

So, what's the difference between a subsequence, a subarray, and a substring?

For instance, in DNA matching, a strand of DNA can be defined as a string over the alphabet

$$\Sigma = \{A, C, G, T\}$$

Formal Specification (Source: CLRS)

Given a sequence X of length m, i.e. $X = \langle x_1, x_2, \ldots, x_m \rangle$ and another sequence Y of length n, i.e. $Y = \langle y_1, y_2, \ldots, y_n \rangle$,

then another sequence $Z = \langle z_1, z_2, \ldots, z_k \rangle$ is a **Common subsequence of X and Y**, if **Z is a subsequence of both X and Y.**

eg.: Let $X = \langle A, B, C, B, D, A, B \rangle$

$Y = \langle B, D, C, A, B, A \rangle$

then,

$Z = \langle B, C, B, A \rangle$ is a common subsequence of X & Y

$\text{len}(Z) = 4$

also, $Z' = \langle B, C, A \rangle$ is another common subsequence.

Is $\langle B, C, A \rangle$ the $LCS(X, Y)$? No. $\langle B, C, B, A \rangle$ is

len: 4

Another sequence $Z'' = \langle B, D, B, A \rangle$ is also $LCS(X, Y)$.

$\text{len}(Z'') = 4$

Another example: Suppose X is a DNA string $\langle CATCGA \rangle$

and Y is another string $\langle GTACCGTCA \rangle$

What is $LCS(X, Y)$? $\langle CTCA \rangle$

Step 01: Check Brute-Force? Exponential time.

Compare all subsequences of X with all subsequences of Y (Verify!). It's Bad!

Step 02: Can we express the problem (LCS) recursively?

Signature _____ No. _____

Date

Prefix String :-

Given a sequence $X = \langle x_1, x_2, \ldots, x_n \rangle$, define the i^{th} prefix of X as $X_i = \langle x_1, x_2, \ldots, x_i \rangle$ where, $0 \leq i \leq m$

(Here X_0 will be an empty string)

eg.,

$X = \langle A\ B\ C\ B\ D\ A\ B \rangle$ (1 2 3 4 5 6 7) then, $X_4 = \langle A\ B\ C\ B \rangle$

$X_0 = \phi$.

Theorem :- Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be two sequences.

Then, let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y.

Then,

1) if $x_m = y_n$, then (z_k must belong to LCS(X, Y)) $z_k = x_m = y_n$.

and $z_{k-1} = LCS(X_{m-1}, Y_{n-1})$.

2) Else if,

$x_m \neq y_n$ and $z_k \neq y_n$

then,

$z_k = LCS(X_m, Y_{n-1})$

3) Else if

$x_m \neq y_n$ and $z_k \neq x_m$

then,

$z_k = LCS(X_{m-1}, Y_n)$

$z_k = (X_0, Y_0) = \phi$
$z_k = (X_m, Y_0) = \phi$
$z_k = (X_0, Y_n) = \phi$

} Base cases of the recursion.

The LCS(X, Y) contains within, the LCS of prefixes of two sequences. Hence, the problem has an optimal substructure.

Signature

RC

No.

Step 03: Overlapping Subproblems?:- Of course!

e.g., let $X = \langle A\ B\ C\ \underset{4}{B} \rangle$ and $Y = \langle B\ D\ C\ A\ B \rangle$
(1 2 3 4) (1 2 3 4 5)

Let's draw the recurrence tree for $LCS(X_m, Y_n)$

$LCS(X_m, Y_n)$

$z_k = x_m = y_n$ (B)

$LCS(X_{m-1}, Y_{n-1})$

and so on....

	1	2	3	4	5
X =	A	B	C	B	
Y =	B	D	C	A	B

$LCS(X_4, Y_5)$

B

$LCS(X_3, Y_4)$

$x_3 \neq y_4$

$LCS(X_3, Y_3)$ — $x_3 = y_3$ C

$LCS(X_2, Y_4)$ — $x_2 \neq y_4$

$LCS(X_2, Y_2)$ — $x_2 \neq y_2$

$LCS(X_2, Y_3)$ — $x_2 \neq y_4$

$LCS(X_1, Y_4)$

$LCS(X_2, Y_1)$ — $x_2 = y_1$ B

$LCS(X_1, Y_2)$ — $x_1 \neq y_2$

$LCS(X_2, Y_2)$ — $LCS(X_1, Y_2)$

Overlapping subproblems!

$LCS(X_1, Y_0)$ = ϕ

$LCS(X_0, Y_2)$

$LCS(X_1, Y_1)$ — $x_1 \neq y_1$

$LCS(X_0, Y_1)$ = ϕ

$LCS(X_1, Y_0)$ = ϕ

Date

Let simplify the problem of LCS(X,Y) to finding the length of LCS(X,Y)

Let,

c[i,j] be the length of $LCS(X_i, Y_j)$.

If $i=0$ or $j=0$, then $c[i,j]=0$ (base-case)

or

$$c[i,j] = \begin{cases} 0, & \text{if } i=0 \text{ or } j=0 \\ c[i-1,j-1]+1, & \text{if } i,j>0 \text{ and } x_i = y_j \\ \max(c[i,j-1], c[i-1,j]), & \text{if } i,j>0 \text{ and } x_i \neq y_j \end{cases}$$

We now solve the problem not recursively but filling the table (bottom-up DP aka tabular approach).
m x n

For our example $X = \underbrace{ABCB}_{m=4}$ and $Y = \underbrace{BDCAB}_{n=5}$

How to fill the table :-

1) Fill the initial cases first (all trivial subproblems)

2) Use any row-major / column-major order:

c[m,n] = 3 (length of LCS)

and

LCS(X,Y) = BCB

(See the table on the next page)

	j	0	1	2	3	4	5
i		y_j	B	D	C	A	B
0	x_i	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	[1]	1	1	1	2
3	C	0	1	1	[2]	2	2
4	B	0	1	1	2	2	[3]

The Lcs(X,Y) is ⟨BCB⟩

c[Lcs(X,Y)] is 3

Signature _____ No. _____