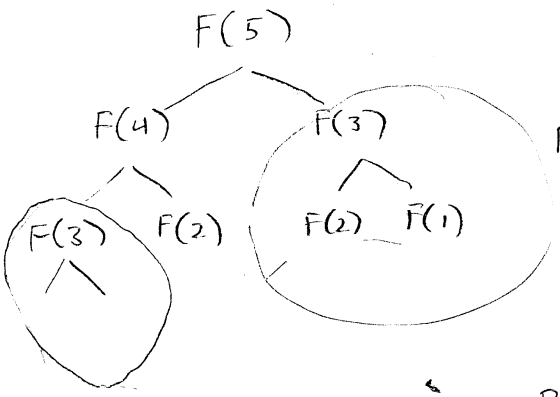
Today
- Dynamic Programming
- P, NP Review

Simple Dynamic Programming Example

Q| Write a program to compute the nth Fibonacci #

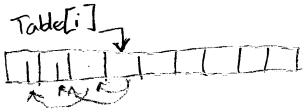
A| Silly program:

F(n) = "1. If n=1 or n=2, output 1
2. Compute F(n-1) = a
3. Compute F(n-2) = b
4. Output a+b "



Bad, computes the same thing many times

Idea: store solutions to subproblems



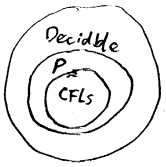
Better Solution:

F(n) = " 1) Let Table[1] := 1 and Table[2] := 2
2) For i = 3, ... n
Table[i] = Table[i-1] + Table[i-2] "

Note: Only actually need to store previous two Table values

Note Also: This shows $L = \{\langle n,k\rangle \mid k \text{ is the } n^{th} \text{ Fib \#}\}$ is in P

Q| Show that all CFLs are in P



A| For a particular CFL L, we must give a poly time algorithm that decides it.

Suppose a CFG G generates it.

Assume G is in Chomsky Normal Form (eg $S \to AB$, $A \to CB$, $C \to 0$, $B \to 1$)

Given $w = \underbrace{w_1 w_2 \cdots}_{A?} \underbrace{w_n}_{B?}$, how do we decide if G generates it?

(eg $w = 0111$)

Idea: For every possible "split" $w_1 \cdots w_K \mid w_{K+1} \cdots w_n$, figure out which variables can generate $w_1 \cdots w_K$, and which can generate $w_{K+1} \cdots w_n$. If we know A can generate the first half, and B the second half, then we know S can generate the whole thing. Apply this idea recursively.

Let $f(w_i \cdots w_j)$ be a function which returns the <u>set</u> of variables that can generate substring $w_i \cdots w_j$. (eg $f(01) = \{A\}$, $f(1) = \{B\}$, $f(11) = \{\}$)

We want to know if $f(w)$ contains S.

To compute $f(w_i \cdots w_j)$ = "
1) If input is a single character a, output all variables Z where there is a rule $Z \to a$
2) For all possible splits $w_i \cdots w_K \mid w_{K+1} \cdots w_j$
   i) Compute $F(w_i \cdots w_K) = \{X_1, X_2 \cdots\}$
   ii) Compute $F(w_{K+1} \cdots w_j) = \{Y_1, Y_2 \cdots\}$
   iii) Output all variables Z where there is a rule $Z \to X_n Y_m$ "

Inefficient as is

Idea: Save previous results in a table.

Table[i,j] stores $F(w_i \cdots w_j)$

First Fill in Table[i,i] for $i = 1 \cdots n$

Then Fill in Table[i,i+1] for $i = 1 \cdots n-1$

" " Table[i,i+2] for $i = 1 \cdots n-2$

⋮

i \ j	1	2	3	4
1	0: C	01: A	011: S	0111: ϕ
2	–	1: B	11: ϕ	111: ϕ
3	–	–	1: B	11: ϕ
4	–	–	–	1: B

Analysis: $O(n^2)$ cells. Each takes $O(n)$ to compute, since we must consider up to n possible "split" points. Total is $O(n^3)$. Thus $L \in P$