CS2102 Analysis and Design of Algorithms

Dynamic Programming

M.Sc. Bryan Gonzales Vega
bgonzales.vega@gmail.com

University of Engineering and Technology

Lecture Content

1. Dynamic Programming
   - Intuition
   - Definition

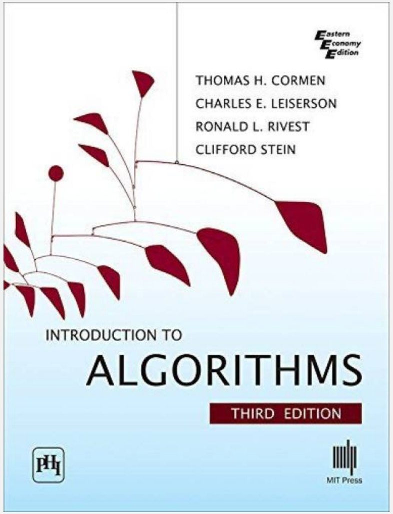
2. Problems
   - 0-1 Knapsack Problem
   - Maximum Contiguous Subarray Sum
   - Coin change

Dynamic Programming

Book Lectures

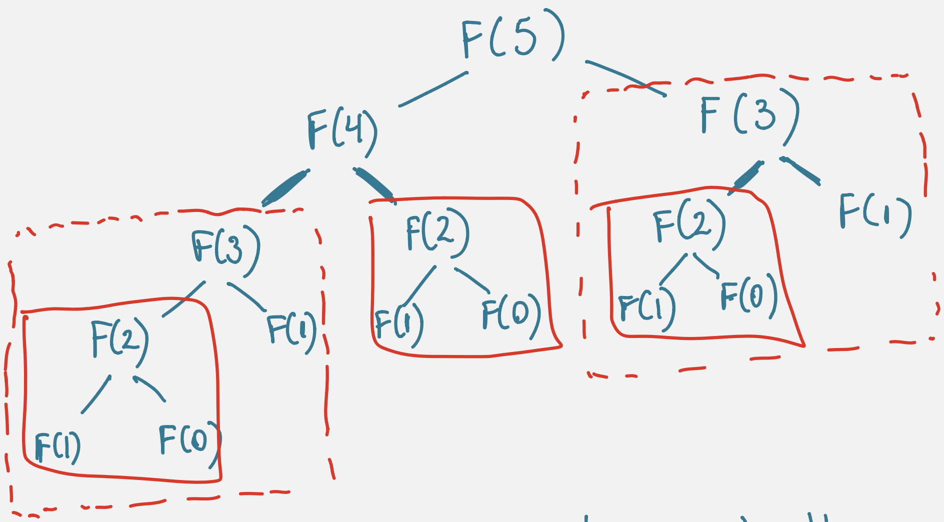
Introduction to Algorithms [Cormen et al., 2009]

- Chap 15: Dynamic Programming



Fibonacci 0 1 1 2 3 5 8 13 21...

$$f(n) = \begin{cases} n, & n = 1 \text{ or } n = 0 \\ f(n-1) + f(n-2), & \text{otherwise} \end{cases} \tag{1}$$



Recursive

```
int fib (int n)
    if (n ≤ 1) return n;
    return fib(n-1) + fib(n-2);
{
```

$O(2^n)$

$\approx 2T(n-1) + 1$

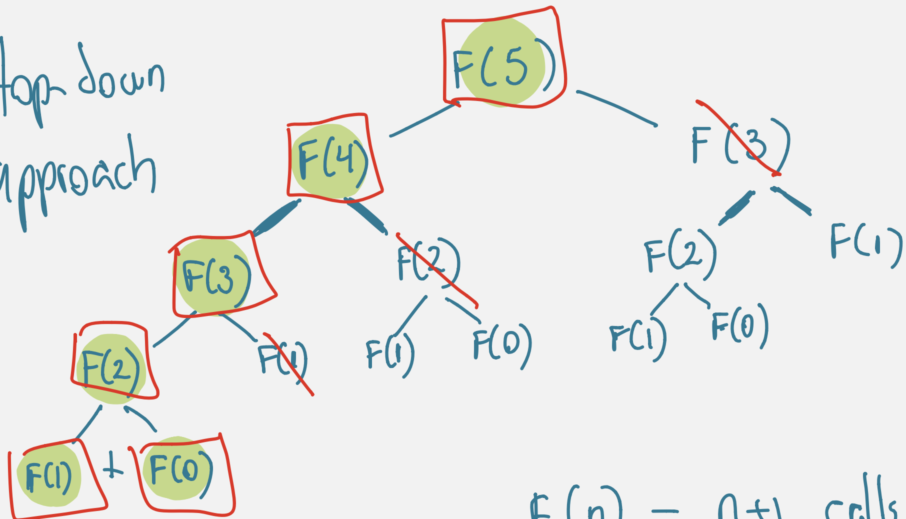
Overlapping subproblems.

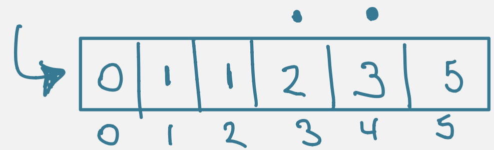
Dynamic Programming Intuition

Fibonacci 0 1 1 2 3 5 8 13 21...

$$f(n) = \begin{cases} n, & n = 1 \text{ or } n = 0 \\ f(n-1) + f(n-2), & \textit{otherwise} \end{cases} \tag{1}$$

top down approach





$F(n) = n+1$ calls

$= O(n)$

memoization

Dynamic Programming Intuition

Fibonacci 0 1 1 2 3 5 8 13 21 ...

$$f(n) = \begin{cases} n, & n = 1 \text{ or } n = 0 \\ f(n-1) + f(n-2), & \textit{otherwise} \end{cases} \tag{1}$$

0	1	2	3	4	5
0	1	1	2	3	5

bottom-up approach

problem -> subproblems

```
int fib(int n){
    F[0]=0, F[1]=1
    for(int i=2; i<=n; ++i){
        F[i]= F[i-1] + F[i-2];
    {
    return F[n];
{
```

$\boxed{\text{tabulation}}$

Directed Acyclic Graph + Shortest Path

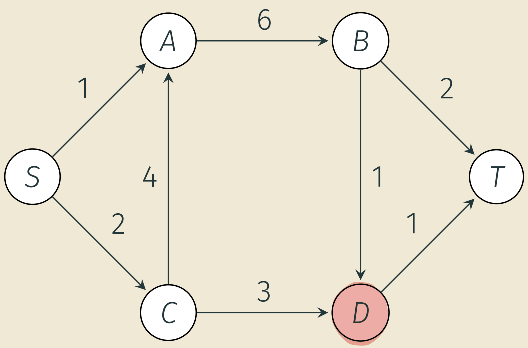


Figure 1: Directed Acyclic Graph (DAG)

the only way to get to D is through its predecessors

$$dist(D) = \min\{dist(B)+1,\ dist(C)+3\}$$

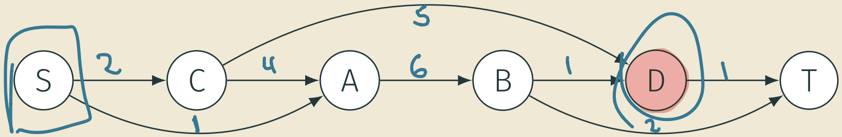


Figure 2: Linearization of Directed Acyclic Graph (topological ordering)

Principle: its nodes can be linearized

```
dist (S) = 0
for ( auto v in vertices.linearized ) {
    dist (v) = min_{(u,v)∈E} { dist (u) + e(u,v) }
}
```

Definition

Dynamic Programming is a paradigm like Divide and Conquer that solves optimization problems and reduces time complexity from exponential to polynomial.

To apply the paradigm some conditions should be met first:

- Overlapping subproblems, subproblems reused several times
- Optimal substructure, optimal solution can be constructed progressively from previous optimal solutions of its subproblems.

Since Dynamic Programming seeks to solve each subproblem only once, we can achieve this using:

- Tabulation or bottom-up approach
- Memoization or top-down approach

Figure 3: Richard Bellman

- Bellman-Ford algorithm
- Curse of dimensionality
- Hamilton-Jacobi-Bellman

Problems

0-1 Knapsack/Backpack Problem

Items	#1	#2	#3	#4
(w) Weights	5kg	3kg	4kg	2kg
(v) Values	60$	50$	70$	30$

Backpack total weight: W
5Kg

weight

	0Kg	1Kg	2Kg	3Kg	4Kg	5Kg
✗ #1	$0	$0	$0	$0	$0	$60
✓ #2	$0	$0	$0	$50	$50	$60
✗ #3	$0	$0	$0	$50	$70	$70
✓ #4	$0	$0	$30	$50	$70	$\boxed{\$80}$

$w_i \leq W$

$$V(i,w) = \begin{cases} \max\{V(i-1,w),\ V_i + V(i-1, W - w_i)\} \\ V(i-1, w), \text{ otherwise} \end{cases}$$

$O(n \cdot m)$

Maximum Contiguous Subarray Sum Problem

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	–3	–25	20	–3	–16	–23	18	20	–7	12	–5	–22	15	–4	7

maximum subarray

start new array

continue subarray

$$[-16 \quad -23 \quad 18 \quad 20 \quad -7 \quad \boxed{12} \quad -5 \quad -22]$$

$$-16 \quad -23 \quad 18 \quad 38 \quad 31 \quad \boxed{43} \quad 38 \quad 16$$

$$m[i] = \max\{A[i],\ A[i] + m[i-1]\}$$

Coin Change Problem

coins = [1, 5, 6, 8]

amount = 11

min # of coins to obtcin an amount.

		0	1	2	3	4	5	6	7	8	9	10	11
X	1	0	1	2	3	4	5	6	7	8	9	10	11
✓	5	0	1	2	3	4	1	2	3	4	5	2	3
✓	6	0	1	2	3	4	1	1	2	3	4	2	2
X	8	0	1	2	3	4	1	1	2	1	2	2	2

$$C(i,j) = \min \begin{cases} C(i-1, j) \\ 1 + C(i, j - c_i) \end{cases}$$

$\rightarrow$ coins = {5, 6}

References i

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009).
Introduction to algorithms.
MIT press.

Cont’d