

Dynamic Programming

03/25/2018

What is DP (Definition)

Dynamic Programming is a method for solving a complex problem by breaking it down into a collection of simpler subproblems. (通过把原问题分解为相对简单的子问题的方式求解复杂问题的方法)

- **Wikipedia**

Dynamic programming is a technique for efficiently implementing a recursive algorithm by storing partial results. The trick is seeing whether the naive recursive algorithm computes the same subproblems over and over and over again.

DP systematically search all possibilities (thus guaranteeing correctness) while storing results to avoid recomputing (thus providing efficiency). By storing the consequences of all possible decisions and using this information in a systematic way, the total amount of work is minimized.

- **S.Skiena**

Caching vs. Computation

Dynamic programming is essentially a tradeoff of space for time. Repeatedly recomputing a given quantity is harmless unless the time spent doing so becomes a drag on performance. Then we are better off storing the results of the initial computation and looking them up instead of recomputing them again.

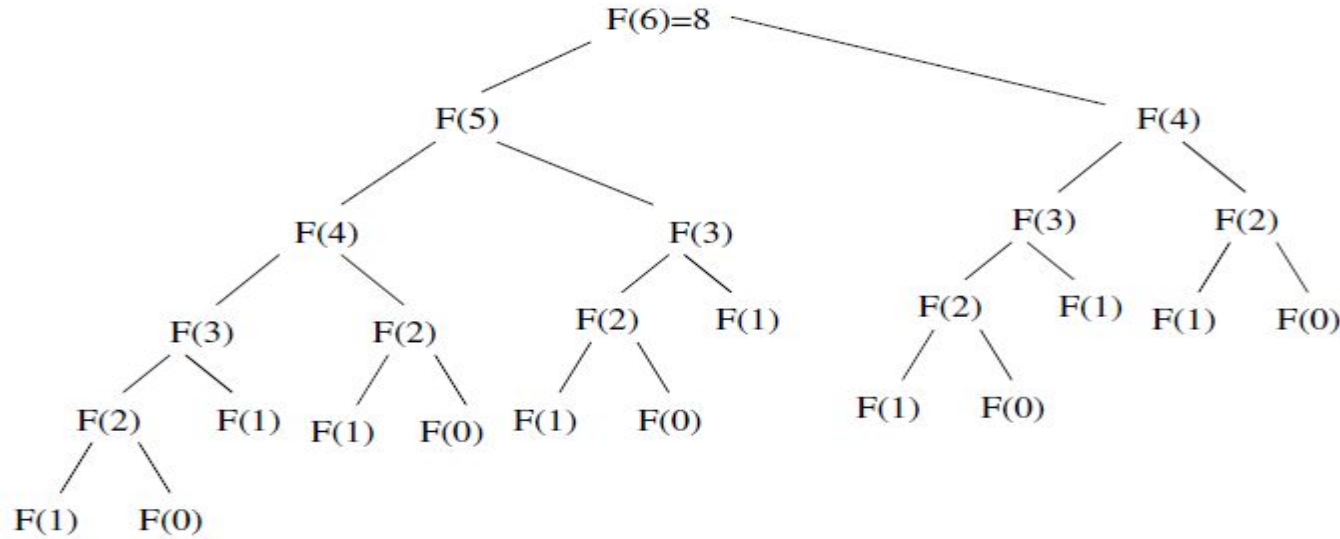
Example: Fibonacci Numbers ($F_n = F_{n-1} + F_{n-2}$)

[Up To 1000th](#)

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765

Let's Code!

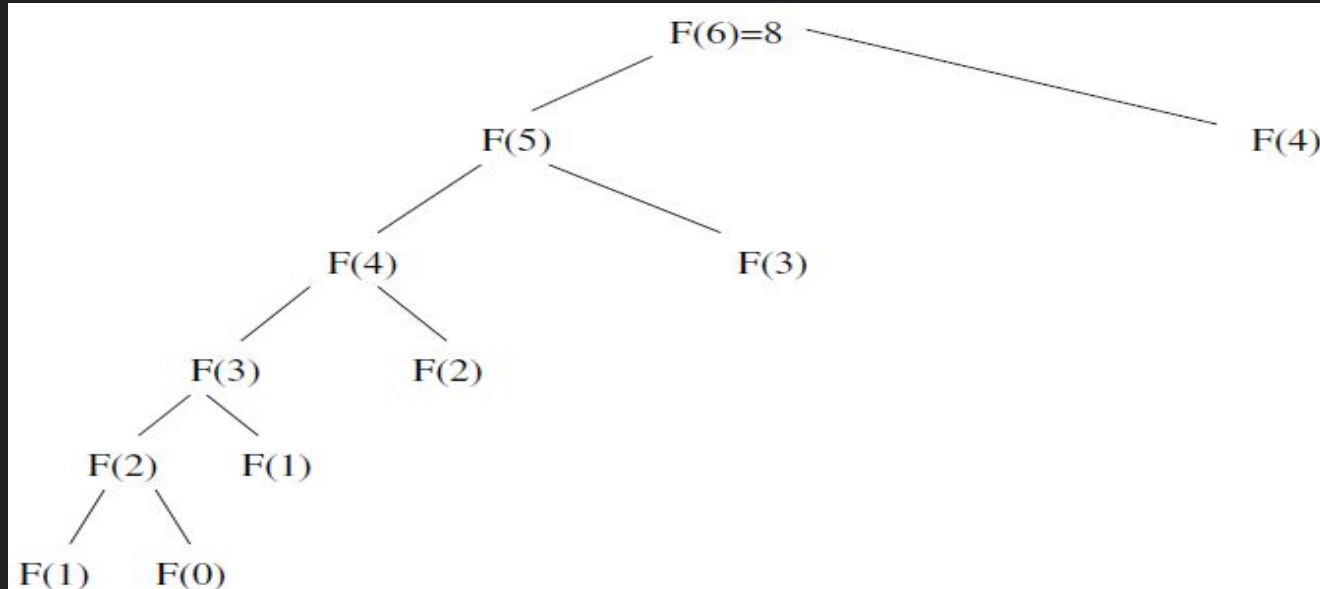
Fibonacci by Recursion



Running Time
 $O(1.6^n)$

How much time does this algorithm take to compute $F(n)$? Since $F_{n+1}/F_n \approx \phi = (1 + \sqrt{5})/2 \approx 1.61803$, this means that $F_n > 1.6^n$. Since our recursion tree has only 0 and 1 as leaves, summing up to such a large number means we must have at least 1.6^n leaves or procedure calls! This humble little program takes exponential time to run!

Fibonacci by Caching (DP)

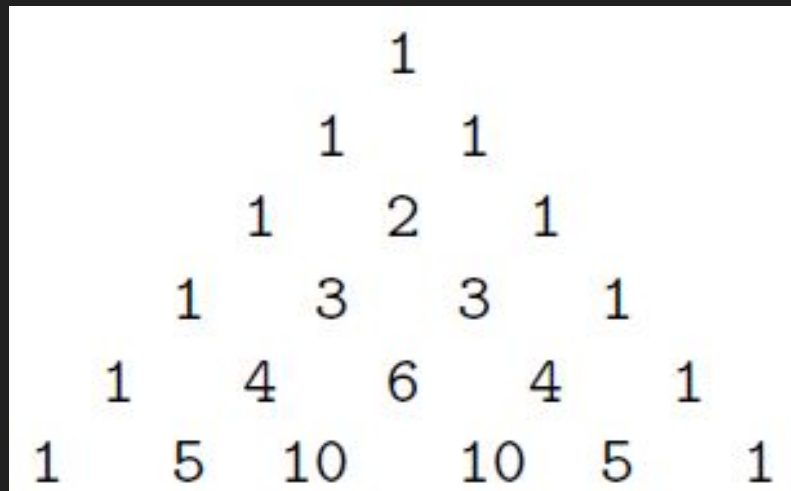


Running Time:
 $O(n)$

Space: $O(n)$
Can we make it better?

Only the left-side calls do computation. The right-side calls find what they are looking for in the cache and immediately return.

拓展: Binomial Coefficients



A Pascal's Triangle diagram showing binomial coefficients. The triangle is centered and has 7 rows. The numbers are arranged in a symmetrical, triangular pattern. The values in each row correspond to the binomial coefficients $\binom{n}{k}$ for n from 0 to 6 and k from 0 to n .

			1			
		1		1		
	1		2		1	
	1	3		3	1	
1	4	6		4	1	
1	5	10	10	5	1	
1						1

Sequence DP

- Longest Increasing Subsequence

$S = \{2, 4, 3, 5, 1, 7, 6, 9, 8\}$; $LIS = \{2, 3, 5, 6, 8\}$

If not using DP..... what can you do....

<u>S</u>	2	4	3	5	1	7	6	9	8
<u>L</u>	1	2	2	3	1	4	4	5	5
<u>P</u>	x	2	2	4/3	x	5	5	6/7	6/7

放松一下 - 爬楼梯问题

You are climbing a stair case. It takes n steps to reach to the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Note: Given n will be a positive integer.

Example 1:

Input: 2

Output: 2

Explanation: There are two ways to climb to the top.

1. 1 step + 1 step
2. 2 steps

Example 2:

Input: 3

Output: 3

Explanation: There are three ways to climb to the top.

1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step

String Matching

Foundation:

- Edit Distance
- Wildcard Matching (通配符匹配)
- Regular Expression Matching (正则表达式匹配)

Variations:

- Leetcode 583 - Delete Operation for Two Strings
- Longest Common Subsequence
- Longest Common Substring

General Steps

There are three steps involved in solving a problem by dynamic programming:

1. Define a State, ex: $f(n)$, $f(i, j)$ - 定义状态
2. Formulate the answer as a recurrence relation - 状态转移方程
3. Initialization - 初始化, 起点
4. Specify an order of evaluation for the recurrence - 计算, 得终点

How to find a Recurrence Relation

- **Edit Distance**

$D(i, j)$: Edit Distance of A's substring $[0, i]$ and B's substring $[0, j]$

$$D(i, j) = \min[D(i-1, j)+1, D(i, j-1) + 1, D(i-1, j-1) + t(i, j)], \quad t(i, j) = 0 / 1$$

- **Longest Increasing Subsequence**

$L(i)$: LIS ends at index i

$$l_i = \max_{0 < j < i} l_j + 1, \text{ where } (s_j < s_i)$$

- **Wildcard Matching**

$$M(i, j) = ???$$

0/1 Knapsack DP (背包问题)

Input description: A set of items $S = \{1, \dots, n\}$, where item i has size s_i and value v_i . A knapsack capacity is C .

Problem description: Find the subset $S' \subset S$ that maximizes the value of $\sum_{i \in S'} v_i$, given that $\sum_{i \in S'} s_i \leq C$; i.e. , all the items fit in a knapsack of size C .

给定一组物品，每种物品都有自己的重量和价格，在限定的总重量内，我们如何选择，才能使得物品的总价格最高。

A backpacker who is constrained by a fixed-size knapsack, and so must fill it only with the most useful and portable items. **Everything has a cost and value, so we seek the most value for a given cost.**

0/1 Knapsack DP (背包问题)

- 不考虑价值, 在 n 个物品中挑选若干物品装入背包, 最多能装多满? 假设背包的容量为 m , 每个物品的大小为 $A[i]$
 - 初始状态是什么?
 - 状态转移方程是什么? $f(i)$ 和 $f(i-1)$ 的关系?
- 考虑价值, 给出 n 个物品的体积 $A[i]$ 和其价值 $V[i]$, 将他们装入一个大小为 m 的背包, 最多能装入的总价值有多大?

0/1 Knapsack DP (背包问题)

- 不考虑价值, 在 n 个物品中挑选若干物品装入背包, 最多能装多满? 假设背包的容量为 m , 每个物品的大小为 $A[i]$

state: $f[i][j]$ “前 i ”个Item, 取出一些, 能否 $\text{sum} == j$

Recurrence Relation: $f[i][j] = f[i-1][j - a[i]]$ or $f[i-1][j]$

Initialize: $f[1...n][0] = \text{true}$; $f[0][1...m] = \text{false}$

answer: 能够使得 $f[n][X]$ 最大的 X ($0 \leq X \leq m$)

0/1 Knapsack DP (背包问题)

- 考虑价值, 给出 n 个物品的体积 $A[i]$ 和其价值 $V[i]$, 将他们装入一个大小为 m 的背包, 最多能装入的总价值有多大?

state: $f[i][j]$ 代表什么?

Recurrence Relation: $f[i][j] = ??$

Initialize: ??

answer: ??

That is it????

