Dynamic Programming

By: Andrew Qi Tang, Ethan Pronev

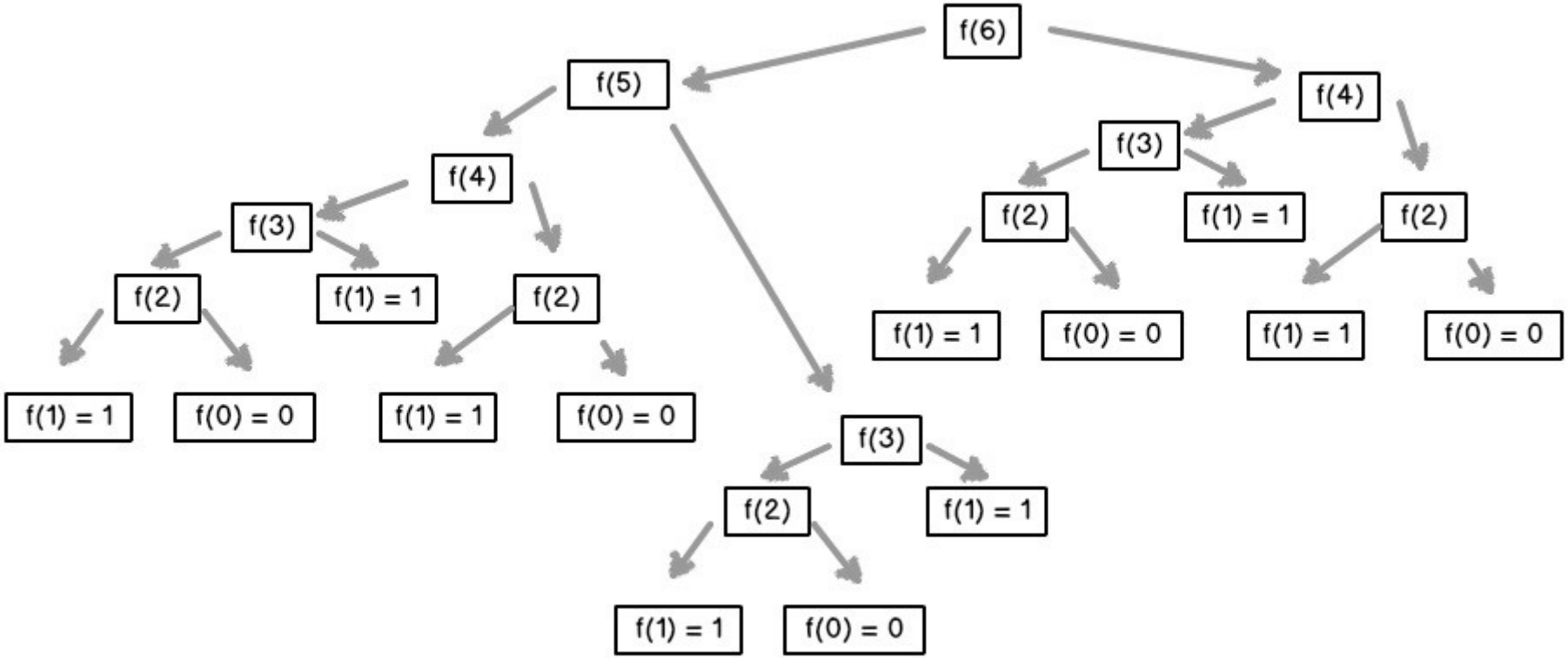
Problem 1: Fibonacci Sequence

Define f(x) = f(x-1) + f(x-2) for x >= 2.

f(0) = 1, f(1) = 1

Find f(k) MOD 1e9+7.

1 <= k <= 100000.



About O(2^N) Operations

What is Dynamic Programming

A mathematical optimization

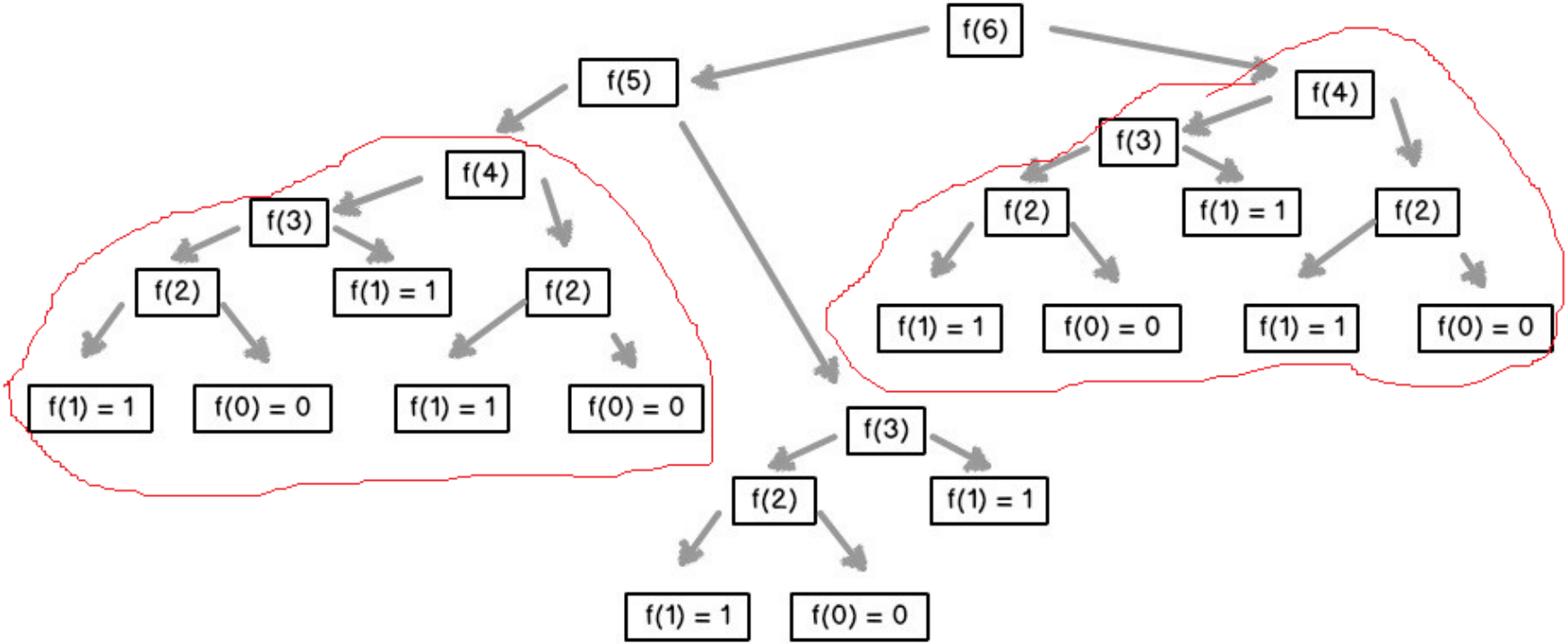
A computer science method

Can change O(2^N) ---> O(N) for the fibonacci sequence

Steps for Dynamic Programming

Needs to have Overlapping Subproblems Property.

1. Define DP States
2. Find the DP Transition
3. Define DP Starting Points and Ending Points
4. Apply Optimizations (Normally not necessary)



We have overlapping subproblems.

1. Define DP States

What you’re going to store

How you’re going to store

For Fibonacci

We will store the K-th Fibonacci number in a 64-bit integer array.

dp[k] = f(k);

2. Find the DP Transition

How are the states related to one another.

For Fibonacci

dp[k] = dp[k-1] + dp[k-2];

3. Define DP Starting And Ending State

The initial values and final values of the DP state.

Sometimes not a single value.

For Fibonacci

Initial States: dp[0] = 1, dp[1] = 1;

Final State: dp[k]

4. Apply Optimizations

Utilise Data Structures to make it faster. (Segment Tree, Deque)

Use Math Expression (Combinatorics, Number Theory Properties)

Apply Common DP Tricks

Try a different DP method or method in general

Propagate the data differently

For Fibonacci

Use Matrix Multiplication O(N) → O(log N) (Not going to be taught)

Problem 2: DP on a Grid

Given a N by M grid.

Every cell (i, j) has a value $a_{i, j}$.

Find maximum value of a path from (1, 1) to (N, M), if you can only go right and up.

N <= 2000, -10^9 <= $a_{i, j}$ <= 10^9

Steps for DP on a Grid

1. Define DP States

   For every row and column store the maximum path to go from (1, 1) to (r, c)

1. Find the DP Transition

   I can come from the top, if I can or I can come from the right if I can.

1. Define DP Starting Points and Ending Points

   Starting state is the maximum path from (1, 1)

   Ending state is maximum path to (N, M)

Problem 3: Counting Summation

Given a number K.

Find the number of ways to represent K as the sum of cute numbers (MOD 1e9+7).

There are M cute numbers given in the input.

K <= 100000, M <= 100

Disclaimer: 1 + 3 is different from 3 + 1

Steps for Counting Summation

1. Define DP States

   For every number n, find the number of ways to represent it as the sum of cute numbers.

1. Find the DP Transition

   For every cute number c: dp[n] += dp[n-c]

1. Define DP Starting Points and Ending Points

   Starting state is n = 0 and ending state is n = K.

Extensions

For DP on a Grid: what if it was a cube? 4 dimensions? K dimensions?

For Counting Summation: what if all numbers from 1 to M are cute.

K <= 10^6, M <= 10^6.

Homework

https://dmoj.ca/problem/ccc12s5

https://dmoj.ca/problem/ccc00s4

https://dmoj.ca/problem/vmss7wc16c5p4

https://dmoj.ca/problem/valday15p2

https://dmoj.ca/problem/ecoo15r1p4

https://dmoj.ca/problem/ccc07s4

Problem 4: 0-1 Knapsack

Given a set of “N”, items where each item has a (value, weight) determine the maximum value you can obtain by making the sum of the weights is less than a given “W”.

N <= 5000, W <= 5000, 1 <= a_i <= 10^9, 1 <= w_i <= 10^9

Sample:

N = 3, W = 6

(2, 3), (4, 5), (3, 2)

Answer: 5 (Take item 1 and item 3)

Let’s Using Greedy Strategy

1. Take the largest values first. Counter Case: N = 3, W = 6 (2, 3), (4, 5), (3, 2) Answer: 5 Greedy: 4	2. Take the smallest weights first. Counter Case N = 2, W = 2 (1, 1), (10, 2) Answer: 10 Greedy: 1	3. Take items by value/weight first. Counter Case N = 3, W = 6 (6, 4), (4, 3), (3, 3) Answer: 7 Greedy: 6 (Works for another problem called fractional knapsack)

Steps For 0-1 Knapsack

1. Define DP State

Dp[n][w] = maximum value if same problem but for the first “n” items and a knapsack with capacity “w”.

1. Define DP Transition

Either we take or don’t take

If weight of the n’s item > w, Dp[n][w] = Dp[n-1][w]

If weight of the n’s item <= w,

Dp[n][w] = max(Dp[n-1][w-n’s weight] + n’s value, Dp[n-1][w]),

Steps For 0-1 Knapsack

3. Base Cases/Terminating Cases

Base Case: Dp[0][j] = 0 where 0 <= 1 <= W.

Terminating Case: Dp[N][W]

Problem 5: Longest Increasing Subsequence LIS

Given an integer array of size N. Find the longest strictly increasing subsequence.

N <= 5000, -10^9 <= a_i <= 10^9.

Sample

N = 7

1 3 2 5 3 7 8

Answer: 5 (1, 2, 5, 7, 8)

Steps For LIS

1. Define DP States

   DP[i] = maximum LIS if contains element “i”

1. Define DP Transition

   DP[i] = max(For every element “j” < “i” and “a_j < a_i”, DP[j])+1

1. Define Base and Ending Cases

   Base case is arguably 0.

   The ending case is the largest value in the Dp array.

   O(N^2) solution. Since O(N) states and O(N) operation per state

Problem 6: Longest Common Subsequence

Given two strings, “s” and “t”, find the length of the longest common subsequence.

“abcabbac”

“cbaac”

Answer: 4 (“cbac”)

Steps For LCS

1. DP[i][j] = maximum subsequence for substring s[1, i], t[1, i].
2. Two cases

   For s[i] != s[j]

   DP[i][j] = max(DP[i-1][j], DP[i][j-1]);

   If character s[i] == s[j]

   DP[i][j] = max(DP[i-1][j-1] + 1, DP[i-1][j], DP[i][j-1]).

1. Base Case DP[0][i] = 0, DP[j][0] for all valid i and j.

   Ending Case DP[N][M] where N is size of s and M is size of M.

What if we want to find the LCS or LIS?

Often needed to:

STORE MORE INFORMATION!!!

WORK BACKWARDS.

Finding the Longest Increasing Subsequence.

Let’s have an array called parent of size N.

Parent[i] will store where they go their answer from.

When done, go backwards. (perhaps recursively or using a while loop)

Terminate when parent[i] = 0.

Think about how to do it for

- LCS string
- Problem 3’s path
- Which items to take from 0-1 Knapsack.

Homework

https://dmoj.ca/problem/dpd

https://dmoj.ca/problem/valday15p2

https://dmoj.ca/problem/dpf

https://dmoj.ca/problem/dmopc13c3p5

https://dmoj.ca/problem/dpe Same problem different constraints as dpd

https://dmoj.ca/problem/dmopc16c1p3

https://dmoj.ca/problem/ccc07s5

https://dmoj.ca/problem/ecoo15r1p4

Game Theory with DP

Nim Game:

- Focus on 2 player games
- No random
- Goal to find who wins the game no matter what the opponent does

Winning State: If at this state, you win the game if played optimally, no matter what the opponent does

Losing State: If at this state, you lose the game no matter what move you do and the opponent plays optimally.

Game Theory with DP

Key Idea:

If a state can reach a losing state, then this state is a winning state. Otherwise, it’s a losing state.

State:

Game states that stores if winning or losing.

Transition:

Possible moves.

Problem 7: CCC 2008 S5 - Nukit

https://dmoj.ca/problem/ccc08s5

Steps for CCC 2008 S5 - Nukit

State:

dp[A][B][C][D] = winning or losing

Transition:

If I use AABDD, ABCD, CCD, BBB, AD can i get to a losing state.

Base Case:

If there exist no 0 As, 0 Bs, 0 Cs, 0 Ds, it’s a losing state.

Code: https://pastebin.com/bSfwH810

Homework

https://dmoj.ca/problem/ccc08s5

https://dmoj.ca/problem/dpk

https://dmoj.ca/problem/tle16c3p4

https://dmoj.ca/problem/ccc15j5

https://dmoj.ca/problem/year2019p2