UNIT 4: Dynamic Programming

Dynamic Programming:

Computing a Binomial Coefficient

Dynamic Programming

- is both a **mathematical optimization method** and a **computer programming method**
- Developed by Richard Bellman in the 1950s
- Applications: in numerous fields, from aerospace engineering, bioinformatics to economics (Example: Ramsey's problem of optimal saving)

Optimal substructure

In computer science,

if a **problem** can be solved optimally by breaking it into **sub-problems** and then recursively finding the **optimal solutions** to the sub-problems, then it is said to have **optimal substructure**.

Overlapping sub-problems

In computer science,

The space of **sub-problems** must be small, that is, any **recursive algorithm** solving the problem should solve the same sub-problems over and over, rather than generating new sub-problems.

Memoization/Memoisation

In computing,

Memoization is an **optimization technique** used primarily to speed up computer programs by storing the results of expensive function calls and returning the cached result when the same inputs occur again.

Dynamic programming strategy

- “programming” refers to technique/planning
- **Two key attributes** that a problem must have in order for dynamic programming to be applicable:
  - optimal substructure and
  - overlapping sub-problems.

- solves each sub-problem only once
- Solves using top-down approach or bottom-up approach

Top-down vs Bottom-up strategy

Top-down approach

- recursive formulation
- overlapping subproblems
- Memoization technique used

Bottom-up approach

- recursive formulation can be reformulated
- overlapping subproblems
- Tabulation method: solve the sub-problems first and then use their solutions to build-on and arrive at solutions to bigger sub-problems. Usually done in a **tabular form by iteratively generating solutions** to bigger and bigger sub-problems by using the solutions to small sub-problems

Example: Fibonacci numbers

Fibonacci numbers

sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Recurrence:

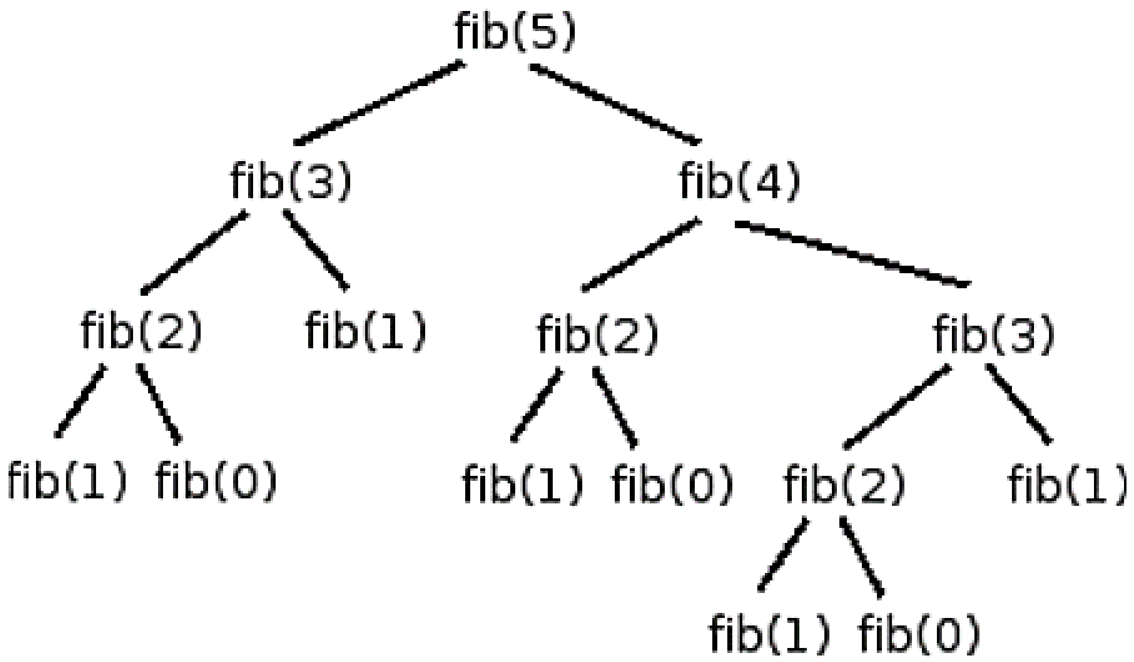
fib(n) = fib(n - 1) + fib(n- 2)   for n > 2

fib(0) = 0

fib(1) = 1

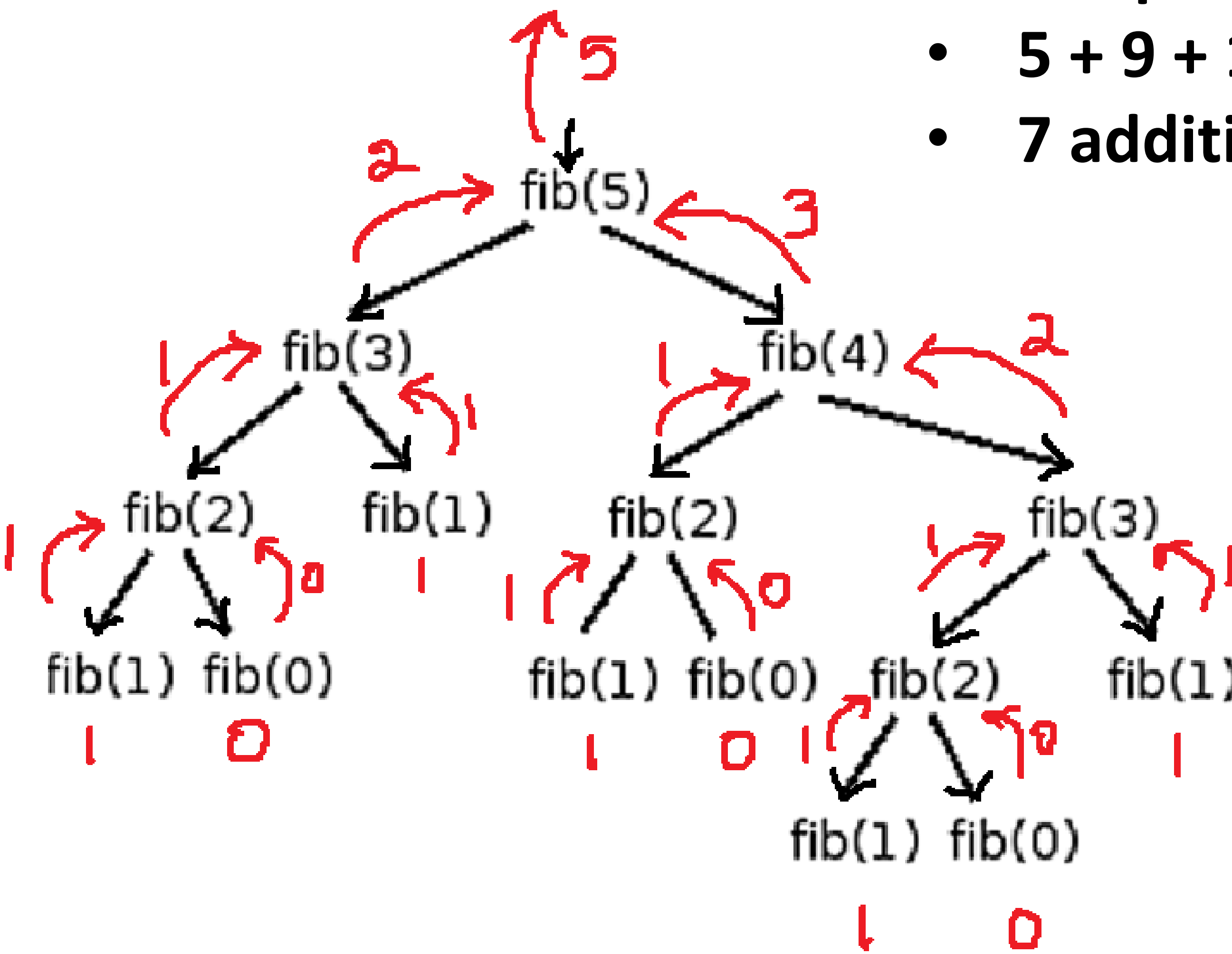
fib(n) = fib(n - 1) + fib(n- 2)   for n > 2
fib(0) = 0, fib(1) = 1

Let's compute fib(5)



Divide and Conquer:

Let's compute fib(5)



To compute fib(5)

- **5 + 9 + 1 = 15 calls**
- **7 additions**

Dynamic programming (Bottom-up) – tabular form:

Let's compute fib(5)

fib(0) = 0

fib(1) = 1

✓ fib(2) = fib(1) + fib(0)

= 0 + 1

= 1

✓ fib(3) = fib(2) + fib(1)

= 1 + 1

= 2

✓ fib(4) = fib(3) + fib(2)

= 2 + 1

= 3

✓ fib(5) = fib(4) + fib(3)

= 3 + 2

= 5

To compute fib(5)

- 4 calls
- 4 additions

Dynamic programming (Top - down) - memoization:

Let's compute fib(5)

fib(5) = fib(4) + fib(3)

3 + 2 = 5

fib(4) = fib(3) + fib(2)

2 + 1 = 3

fib(3) = fib(2) + fib(1)

1 + 1 = 2

fib(2) = fib(1) + fib(0)

= 1 + 0

= 1

fib(0) = 0
fib(1) = 1
fib(2) = 1
fib(3) = 2
fib(4) = 3
fib(5) = 5

To compute fib(5)

- 3 + 1 calls
- 4 additions

Divide and Conquer
vs
Dynamic Programming

Divide and Conquer strategy

- Three steps: Divide, Conquer, Combine
- Recursive
- All subproblems are independent
- Solves all the subproblems
- Follows top-down approach

Dynamic programming strategy

- Four steps:
    1. Characterize the structure of optimal solutions.
    2. Recursively defines the values of optimal solutions.
    3. Compute the value of optimal solutions in a Bottom-up minimum.
    4. Construct an Optimal Solution from computed information.

- recursive or non-recursive
- subproblems are interdependent
- solves subproblems only once and then stores in the table
- follows top-down, bottom-up approach

Example:
Computing a Binomial Coefficient

Computing a Binomial Coefficient

- standard example of applying dynamic programming to a non-optimization problem

(elementary combinatorics)

binomial coefficient, C(n, k) or $\binom{n}{k}$

= the number of combinations (subsets) of k

elements from an n-element set ($0 \le k \le n$)

Two properties from binomial formula leads to the definition of binomial coefficients:

$$(a+b)^n = C(n,0)a^n + \cdots + C(n,k)a^{n-k}b^k + \cdots + C(n,n)b^n.$$

$$C(n,k) = C(n-1,k-1) + C(n-1,k) \quad \text{for } n > k > 0$$

$$C(n,0) = C(n,n) = 1$$

C(n, k) is defined in terms of the smaller and overlapping problems of computing

- record the values of the binomial coefficients in a table of **n + 1** rows and **k + 1** columns, numbered from 0 to n and from 0 to k, respectively

	0	1	2	...	k-1	k
0	1					
1	1	1				
2	1		1			
	1			1		
...						
k-1	1				1	
k						1
...	1					
n-1	1				C(n-1, k-1)	C(n-1, k)
n	1					**C(n, k)**

$$C(n, k) = C(n-1, k-1) + C(n-1, k) \quad \text{for } n > k > 0$$

$$C(n, 0) = C(n, n) = 1$$

C(6, 4)

	0	1	2	3	4
0	1				
1	1	1			
2	1	2	1		
3	1	3	3	1	
4	1	4	6	4	1
5	1	5	10	10	5
6	1	5	15	20	15

```
ALGORITHM  Binomial(n, k)
    //Computes C(n, k) by the dynamic programming algorithm
    //Input: A pair of nonnegative integers n ≥ k ≥ 0
    //Output: The value of C(n, k)
    for i ← 0 to n do
        for j ← 0 to min(i, k) do
            if j = 0 or j = i
                C[i, j] ← 1
            else C[i, j] ← C[i − 1, j − 1] + C[i − 1, j]
    return C[n, k]
```

```
ALGORITHM  BinomCoeff(n, k)
    if k = 0 or k = n return 1
    else return BinomCoeff(n − 1, k − 1)+BinomCoeff(n − 1, k)
```

Analysis: Binomial Coefficient

- Input size: **n and k**
- Basic operation: **Addition**
- Let **A (n, k)** be the total number of additions made in computing C(n, k).

(Note: First k + 1 rows of the table form a triangle while the remaining n - k rows form a rectangle)

$$
\begin{aligned}
A(n, k) &= \sum_{i=1}^{k} \sum_{j=1}^{i-1} 1 + \sum_{i=k+1}^{n} \sum_{j=1}^{k} 1 \\
&= \sum_{i=1}^{k} (i-1) + \sum_{i=k+1}^{n} k \\
&= \frac{(k-1)k}{2} + k(n-k) \\
&\in \Theta(nk)
\end{aligned}
$$

Puzzles/Common data structure problems that can be solved using Dynamic programming

Longest/Shortest sequence problems

- Longest Common Subsequence | LCS Length, Finding all LCS
- Shortest Common Supersequence | SCS Length, Finding all SCS
- Longest Repeated Subsequence Problem
- Implement Diff Utility
- Longest Increasing Subsequence
- Longest Bitonic Subsequence
- Increasing Subsequence with Maximum Sum
- Longest Alternating Subsequence Problem

String problems

- Longest Common Substring problem
- Longest Palindromic Subsequence
- Longest Repeated Subsequence Problem
- Count number of times a pattern appears in given string as a subsequence
- Word Break Problem
- Wildcard Pattern Matching
- Longest Alternating Subsequence Problem
- Check if given string is interleaving of two other given strings

Matrix problems

- Find size of largest square sub-matrix of 1’s present in given binary matrix
- Matrix Chain Multiplication
- Find the minimum cost to reach last cell of the matrix from its first cell
- Find longest sequence formed by adjacent numbers in the matrix
- Count number of paths in a matrix with given cost to reach destination cell
- Find Maximum Sum Submatrix in a given matrix
- Find maximum sum of subsequence with no adjacent elements
- Collect maximum points in a matrix by satisfying given constraints
- Calculate sum of all elements in a sub-matrix in constant time

Graph problems

- The Levenshtein distance (Edit distance) problem
- Single-Source Shortest Paths — Bellman Ford Algorithm
- All-Pairs Shortest Paths — Floyd Warshall Algorithm

Optimization, Combinatorial problems

- 0–1 Knapsack problem
- Maximize the Value of an Expression
- Minimum Sum Partition Problem
- Rod Cutting Problem
- Maximum Product Rod Cutting
- Coin change-making problem (unlimited supply of coins)
- Coin Change Problem (Total number of ways to get the denomination of coins)
- Find Optimal Cost to Construct Binary Search Tree
- Count total possible combinations of N-digit numbers in a mobile keypad

Decision making and other problems

- Subset Sum Problem
- Partition problem
- Find minimum cuts needed for palindromic partition of a string
- 3-Partition Problem
- Find all N-digit binary strings without any consecutive 1's
- Calculate size of the largest plus of 1's in binary matrix
- Total possible solutions to linear equation of k variables
- Find Probability that a Person is Alive after Taking N steps on an Island
- Maximum Subarray Problem (Kadane's algorithm)
- Pots of Gold Game using Dynamic Programming
- Maximum Length Snake Sequence