Dynamic Programming

Invented by American mathematician Richard Bellman in the 1950s to solve optimization problems and later assimilated by CS. "Programming" here means "planning"

D*ynamic Programming* is a powerful algorithm design technique for solving problems that

- Appear to be exponential but have a poly solution with DP
- In many cases are optimization problems (min/max)
- Defined by or formulated as recurrences with overlapping subproblems
- Optimal solution to a problem contains optimal solutions to subproblems.

Dynamic Programming

- Like divide and conquer, DP solves problems by combining solutions to subproblems.
- Unlike divide and conquer, subproblems are not unique.
  - Subproblems may share subsubproblems,
  - However, solution to one subproblem may not affect the solutions to other subproblems of the same problem.
- Key: Determine structure of optimal solutions

5 Steps to DP

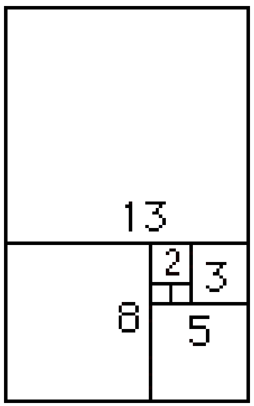
1. Define subproblems
2. Guess part of the solution
3. Relate subproblem solutions
4. Recurse + memoize or Build a DP bottom-up table.
5. Solve original problem

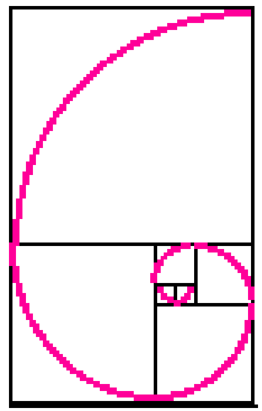
DP Examples

- Fibonacci
- Binomial Coefficients
- Longest Common Subsequence
- Longest Increasing Subsequence
- Knapsack
- Shortest Path
- Chain Matrix Multiplication
- Edit Distance
- Rod Cutting
- Optimal BST

Fibonacci Sequence

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

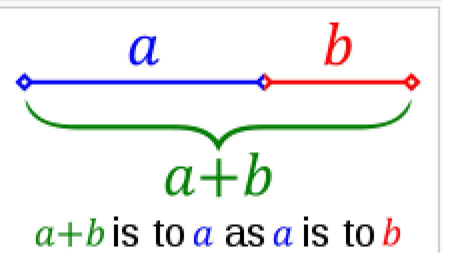




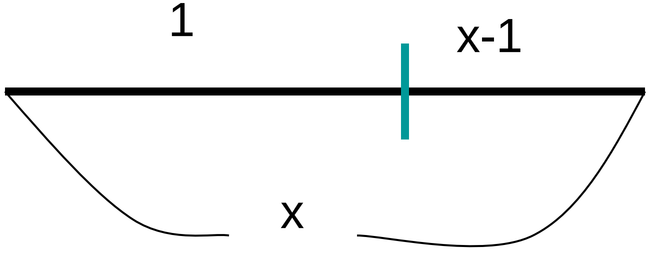
Fibonacci Number and Golden Ratio

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

$$\begin{cases} f_n = 0 & \text{if } n = 0 \\ f_n = 1 & \text{if } n = 1 \\ f_n = f_{n-1} + f_{n-2} & \text{if } n \geq 2 \end{cases}$$



$$\lim_{n\to\infty} \frac{f_n}{f_{n-1}} = \frac{1+\sqrt{5}}{2} = \text{Golden Ratio} = \phi = 1.61803..$$



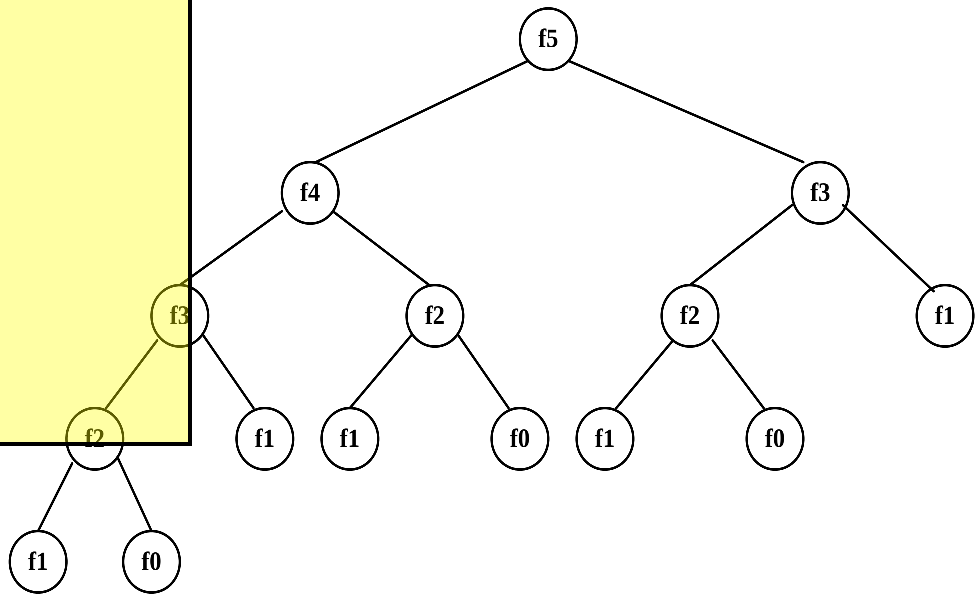
$$\frac{x}{1} = \frac{1}{x-1}$$

$$x^2 - x - 1 = 0$$

$$x = \frac{1+\sqrt{5}}{2}$$

Naive Recursive Algorithm

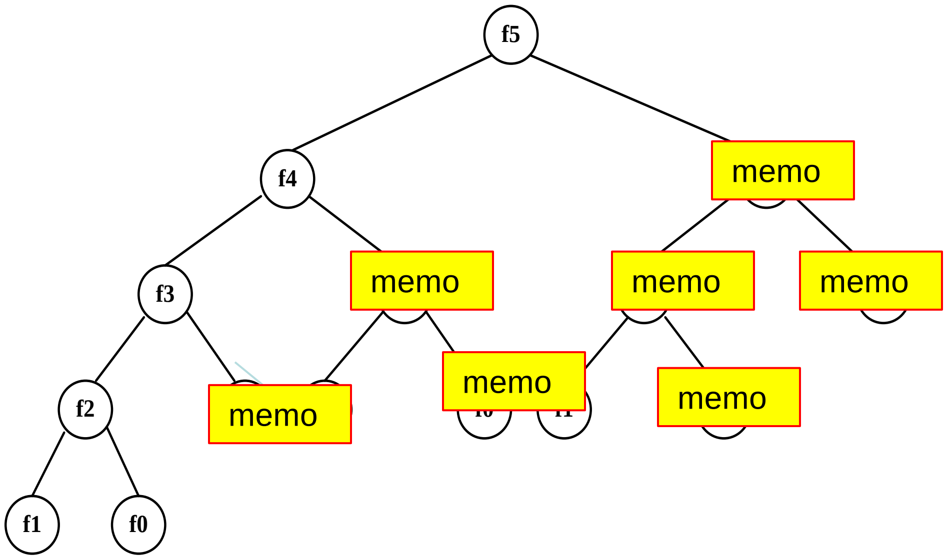
```
  fib (n) {
    if (n = 0) {
        return 0;
    } else if (n = 1) {
        return 1;
    } else {
        return fib(n-1) + fib(n-2);
    }
}
```



- Solved by a recursive program
- Much replicated computation is done.
- Running time $\Theta(\phi^n)$ - exponential

Memoized DP Algorithm

```
memo = { }
fib (n) {
    if (n in memo) { return memo[n] }
    if (n <= 1) {
        f =  n;
     } else {
        f =  fib(n-1) + fib(n-2);
    }
    memo[n] = f;
    return f
}
```



- fib(k) only recurses the first time called only n nonmemoized calles
- Memorized calls "free" $\Theta(1)$.
- Time = #subproblems * time/subproblem

  $= n * \Theta(1)$
- Running time $\Theta(n)$ - linear

Bottom-up DP Algorithm

```
fib = { }
fib[0] = 0;
fib[1] = 1;
for k = 2 to n
      fib[k] =  fib[k-1] + fib[k-2];
return fib[n]
```

- Same as memoized DP with recursion “unrolled” into iteration.
- Practically faster since no recursion
- Analysis is more obvious
- Running time $\Theta(n)$ - linear

A Basic Idea of Dynamic Programming

- DP = recursion + memoization
  - Memoize = remember and reuse solutions to subproblems
- Botton-Up Method stores all values in a table

Binomial Coefficient/Combinations C(n, k)

- the number of ways can you select k lottery balls out of n
- the number of birth orders possible in a family of n children where k are sons
- the number of acyclic paths connecting 2 corners of an $k \times (n-k)$ grid
- the coefficient of the $a^k b^{n-k}$ term in the polynomial expansion of $(a + b)^n$

$$C(n,k) \equiv \binom{n}{k} \equiv \frac{n!}{k!(n-k)!}$$

Example: Combinations

While easy to define, a binomial coefficient is difficult to compute

6 number lottery with 49 balls → 49!/6!43!

49! = 608,281,864,034,267,560,872,252,163,321,295,376,887,552,831,379,210,240,000,000,000

could try to get fancy by canceling terms from numerator & denominator

– can still can end up with individual terms that exceed integer limits

A computationally easier approach makes use of the following recursive relationship

$$\binom{n}{k} \equiv \binom{n-1}{k-1} + \binom{n-1}{k}$$

e.g., to select 6 lottery balls out of 49, partition into:

selections that include 1
(must select 5 out of remaining 48)
+
selections that don't include 1
(must select 6 out of remaining 48)

Recursive Binomial Coefficient

could use straight divide & conquer to compute based on this relation

```
/** using divide-and-conquer
 * Calculates n choose k
 *   n the total number to choose from (n > 0)
 *   k the number to choose (0 <= k <= n)
 * */
int Binomial(int n, int k) {
    if (k == 0 || n == k) {
        return 1;
    }
    else {
        return Binomial(n-1, k-1) + Binomial(n-1, k);
    }
}
```

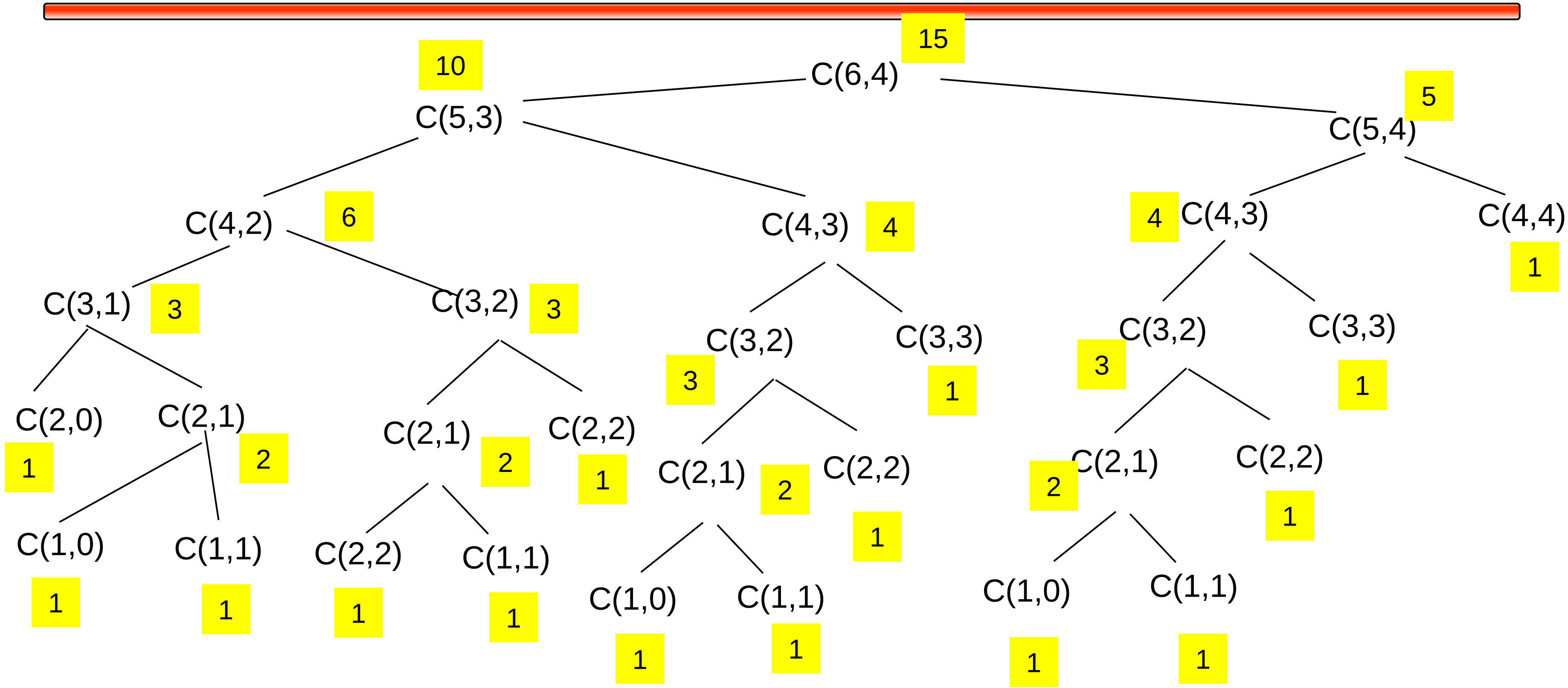
however, this will take a long time or exceed memory due to redundant work

$$\binom{49}{6}$$

$$\binom{48}{5} + \binom{48}{6}$$

$$\binom{47}{4} + \binom{47}{5} \qquad \binom{47}{5} + \binom{47}{6}$$

Binomial Coefficient Tree



$$T(n, k) = T(n-1,k-1) + T(n-1, k) + 1$$

$$T(k,0) = 1, \quad T(k,k) = 1$$

Binomial Coefficient Recurrence

$T(n, k) = T(n-1,k-1) + T(n-1, k) + 1$

$T(n,0) = 1, \quad T(n,n) = 1$

$T(n, k) = T(n-1,k-1) + T(n-1, k) + 1 < 2\,T(n-1) + 1$

. . . .

$O(2^n)$

Computing a Binomial Coefficient by DP

Recurrence: $C(n,k) = C(n-1,k) + C(n-1,k-1)$ for $n > k > 0$

$C(n,0) = 1$, $C(n,n) = 1$ for $n \geq 0$

	K = 0	1	2	3	4	5	6
N = 0	1						
1	1	1					
2	1		1				
3	1			1			
4	1				1		
5	1					1	
6	1						1

Computing a Binomial Coefficient by DP

Recurrence: $C(n,k) = C(n-1,k) + C(n-1,k-1)$ for $n > k > 0$

$C(n,0) = 1$, $C(n,n) = 1$ for $n \geq 0$

	K = 0	1	2	3	4	5	6
N = 0	1						
1	1	1					
2	1	1+1=2	1				
3	1			1			
4	1				1		
5	1					1	
6	1						1

Computing a Binomial Coefficient by DP

Recurrence: $C(n,k) = C(n-1,k) + C(n-1,k-1)$ for $n > k > 0$

$C(n,0) = 1$, $C(n,n) = 1$ for $n \geq 0$

	K = 0	1	2	3	4	5	6
N = 0	1						
1	1	1					
2	1	2	1				
3	1	3		1			
4	1				1		
5	1					1	
6	1						1

Computing a Binomial Coefficient by DP

Recurrence: $C(n,k) = C(n-1,k) + C(n-1,k-1)$ for $n > k > 0$

$C(n,0) = 1$, $C(n,n) = 1$ for $n \geq 0$

	K = 0	1	2	3	4	5	6
N = 0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4	1				1		
5	1					1	
6	1						1

Computing a Binomial Coefficient by DP

Recurrence: $C(n,k) = C(n-1,k) + C(n-1,k-1)$ for $n > k > 0$

$C(n,0) = 1$, $C(n,n) = 1$ for $n \geq 0$

	K = 0	1	2	3	4	5	6
N = 0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4	1	4			1		
5	1					1	
6	1						1

Computing a Binomial Coefficient by DP

Recurrence: $C(n,k) = C(n-1,k) + C(n-1,k-1)$ for $n > k > 0$

$C(n,0) = 1$, $C(n,n) = 1$ for $n \geq 0$

	K = 0	1	2	3	4	5	6
N = 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	1	
6	1	6	15	20	15	6	1

DP Algorithm Binomial Coefficient

```
Binomial (n,k)
// Computes C(n,k) by DP
// 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]
```

Running time: $\Theta(nk)$

Space efficiency: $\Theta(nk)$

Recurrence: $C(n,k) = C(n-1,k) + C(n-1,k-1)$ for $n > k > 0$

$C(n,0) = 1$, $C(n,n) = 1$ for $n \geq 0$