Design and Analysis of Algorithms

Dynamic Programming

Reetinder Sidhu

Department of Computer Science and Engineering

Design and Analysis of Algorithms

Dynamic Programming

Reetinder Sidhu

Department of Computer Science and Engineering

- Dynamic Programming
  - **Computing a Binomial Coefficient**
  - The Knapsack Problem
  - Memory Functions
  - Warshall's and Floyd's Algorithms
- Limitations of Algorithmic Power
  - Lower-Bound Arguments
  - Decision Trees
  - P, NP, and NP-Complete, NP-Hard Problems
- Coping with the Limitations
  - Backtracking
  - Branch-and-Bound

Concepts covered

- Dynamic Programming
  - Introduction
  - Fibonacci numbers
  - Binomial Coefficients

Introduction

Dynamic Programming is a general algorithm design technique for solving problems defined by recurrences with overlapping subproblems

- Invented by American mathematician Richard Bellman in the 1950s to solve optimization problems and later assimilated by CS
- “Programming” here means “planning”
- Main idea:
  - set up a recurrence relating a solution to a larger instance to solutions of some smaller instances
  - solve smaller instances once
  - record solutions in a table
  - extract solution to the initial instance from that table

Dynamic Programming
Example: Fibonacci Numbers

- Recall definition of Fibonacci numbers:

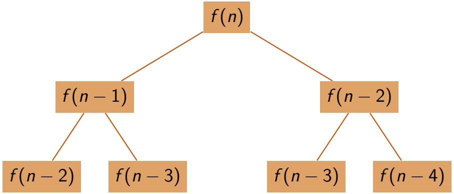
$$
\begin{aligned}
f(n) &= f(n-1) + f(n-2)\\
f(0) &= 0\\
f(1) &= 1
\end{aligned}
$$

Dynamic Programming
Example: Fibonacci Numbers

- Recall definition of Fibonacci numbers:

$$
\begin{aligned}
f(n) &= f(n-1) + f(n-2) \\
f(0) &= 0 \\
f(1) &= 1
\end{aligned}
$$

- Computing the n^{th} Fibonacci number recursively (top-down):



Dynamic Programming
Example: Fibonacci Numbers

Computing the nth Fibonacci number using bottom-up iteration and recording results:

$$
\begin{aligned}
f(0) &= 0 \\
f(1) &= 1 \\
f(2) &= 0 + 1 = 1 \\
f(3) &= 1 + 1 = 2 \\
f(4) &= 1 + 2 = 3 \\
&\vdots
\end{aligned}
$$

Example: Fibonacci Numbers

Computing the nth Fibonacci number using bottom-up iteration and recording results:

$$
\begin{aligned}
f(0) &= 0 \\
f(1) &= 1 \\
f(2) &= 0 + 1 = 1 \\
f(3) &= 1 + 1 = 2 \\
f(4) &= 1 + 2 = 3 \\
&\vdots
\end{aligned}
$$

Efficiency:

- time: $\Theta(n)$
- space: $\Theta(n)$ or $\Theta(1)$

Algorithm Examples

- Computing a binomial coefficient
- Warshall’s algorithm for transitive closure
- Floyd’s algorithm for all-pairs shortest paths
- Constructing an optimal binary search tree
- Some instances of difficult discrete optimization problems:
  - traveling salesman
  - knapsack

Binomial Coefficient

- Binomial coefficients are coefficients of the binomial formula:

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

Binomial Coefficient

- Binomial coefficients are coefficients of the binomial formula:

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

- Recurrence:

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

$$C(n,0) = 1, C(n,n) = 1 \quad \text{for } n \geq 0$$

Dynamic Programming

Binomial Coefficient

- Binomial coefficients are coefficients of the binomial formula:

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

- Recurrence:

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

$$C(n,0) = 1, C(n,n) = 1 \quad \text{for } n \geq 0$$

- Value of C(n,k) can be computed by filling a table:

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

Dynamic Programming

Binomial Coefficient Algorithm

Dynamic Programming Binomial Coefficient Algorithm

```
procedure BINOMIAL(n, k)
    ▷ Input: Integers n ≥ 0,  k ≥ 0
    ▷ Output: C(n, k)
    for i ← 0 to n do
        for j ← 0 to min(i, k) do
            if j=0  or  j=i then
                C(i, j) ← 1
            else C[i, j] = C[i − 1, j] + C[i − 1, j − 1]
    return C[n, k]
```

DYNAMIC PROGRAMMING
Binomial Coefficient Algorithm

Dynamic Programming Binomial Coefficient Algorithm

```
procedure BINOMIAL(n, k)
    ▷ Input: Integers n ≥ 0, k ≥ 0
    ▷ Output: C(n, k)
    for i ← 0 to n do
        for j ← 0 to min(i, k) do
            if j=0 or j=i then
                C(i, j) ← 1
            else C[i, j] = C[i − 1, j] + C[i − 1, j − 1]
    return C[n, k]
```

- Time: $\Theta(nk)$
- Space: $\Theta(nk)$

Think About It

- What does dynamic programming have in common with divide-and-conquer? What is a principal difference between them?

Think About It

- What does dynamic programming have in common with divide-and-conquer? What is a principal difference between them?
- The coin change problem does not have an optimal greedy solution in all cases (*ex:* coins 1,20,25 and amount 40). Is there a dynamic programming based algorithm that can solve all cases of the coin change problem?