

CS 212 – DESIGN AND ANALYSIS OF ALGORITHMS

LECTURE 5
(DYNAMIC PROGRAMMING)

George Washington University

Fibonacci Numbers

2

- Defined as
 - $f(1) = f(2) = 1$
 - $f(n) = f(n-1) + f(n-2)$
- So, a simple recursive program can be:


```
function fib(int n) {
    if (n <= 2) {
        return 1;
    }
    return f(n - 1) + f(n - 2);
}
```
- What is the time complexity of this algorithm?

Lecture 5 CS212 - Arora

A quote to remember

3

"I have only proven the solution correct – I haven't actually tested it."

-- Hugely inspired from Edsger Dijkstra

Lecture 5 CS212 - Arora

Recursive Program Output

4

Number	Fibonacci	Recursive	
5	5	5	0
10	55	55	0
15	610	610	0
20	6765	6765	1
25	75025	75025	1
30	832040	832040	8
35	9227465	9227465	83
40	102334155	102334155	922
45	1134903170	1134903170	10233
50	12586269025	12586269025	113770

Lecture 5 CS212 - Arora

Recursive Program Output

5

Lecture 5 CS212 - Arora

Fibonacci Numbers – Alternate Solution

6

```
long a[] = new long[n + 1];
a[1] = a[2] = 1;
for (int i = 3; i <= n; i++) {
    a[i] = a[i - 1] + a[i - 2];
}
return a[n];
```

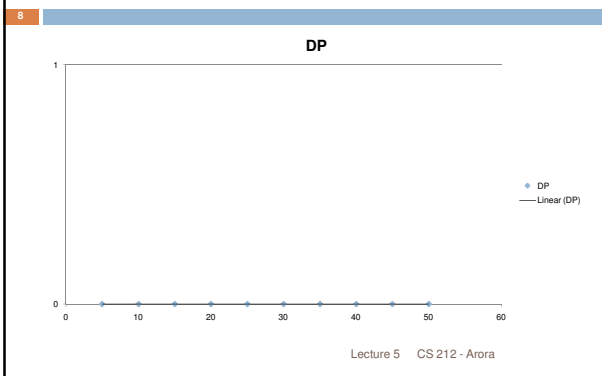
Lecture 5 CS212 - Arora

Dynamic Program Output

Number	Fibonacci	DP
5	5	0
10	55	0
15	610	0
20	6765	0
25	75025	0
30	832040	0
35	9227465	0
40	102334155	0
45	1134903170	0
50	12586269025	0

Lecture 5 CS 212 - Arora

Dynamic Program Trend line



Dynamic Programming (General formulation)

- A computation/optimization technique
 - Computes its solution bottom up by synthesizing them from smaller subsolutions
 - Stores the results of subsolutions to avoid recomputation of sub problems
 - Very useful technique when:
 - Optimal substructure
 - Overlapping subproblems
- Lecture 5 CS 212 - Arora

Optimal Substructure

- A problem is said to have optimal substructure if:
 - **(Definition 1)** An optimal solution can be constructed efficiently from optimal solutions to its subproblems
 - **(Definition 2)** The subsolutions of an optimal solution of the problem are themselves optimal solutions for their subproblems.
 - **Example:** If $a, x_1, x_2, \dots, x_n, b$ is a shortest path from node a to node b in a graph, then the portion of x_i to x_j on that path is a shortest path from x_i to x_j as well.
- Lecture 5 CS 212 - Arora

Overlapping Subproblems

- fib(5)
 - fib(4) + fib(3)
 - (fib(3) + fib(2)) + fib(3)
 - fib(4) and fib(3) are not disjoint, but overlap each other
 - In fact fib(3) is completely within fib(4)
- Lecture 5 CS 212 - Arora

Overlapping Subproblems (Cont.)

Carl Hamppe vs Philipp Meitner
 "The Immortal Draw" (game of the day Jan-31-97)
 Vienna s8/2 - Vienna Game: Hamppe-Meitner Variation (C63) - 1/2-1/2

Lecture 5 CS 212 - Arora

DP – Trivia Info

13

- Term originally used in the 1940s by Richard Bellman to describe the process of solving problems where one needs to find the best decisions one after another.
- By 1953, he had refined this to the modern meaning, which refers specifically to nesting smaller decision problems inside larger decisions.
- Bellman equation is a central result of dynamic programming which restates an optimization problem in recursive form.

Lecture 5 CS 212 - Arora

Dynamic Programming Template

14

- Develop a mathematical notation that can express any solution and subsolution for the problem at hand.
- Prove that the Optimal Substructure (Principle of Optimality) holds.
- Develop a recurrence relation that relates a solution to its subsolutions, using the math notation of step 1.
- Write an algorithm to compute the recurrence relation.

Lecture 5 CS 212 - Arora

Matrix Chain Problem

15

- Input: n matrices A_1, A_2, \dots, A_n of dimensions $r_1 \times c_1, r_2 \times c_2, \dots, r_n \times c_n$, respectively.
 - Obviously $c_i = r_{i+1}$ for $1 < i < n$
- Goal: To compute the matrix product $A_1 A_2 \dots A_n$
- Problem: In what order should $A_1 A_2 \dots A_n$ be multiplied so that it would take the minimum number of computations to derive the product.
- Note that when multiplying two matrices of sizes, $(a \times b)$ and $(b \times c)$ the cost = $a \cdot b \cdot c$.

Lecture 5 CS 212 - Arora

Example

16

- Say given 3 matrices:
 - A_1 of dimensions 3×5
 - A_2 of dimensions 5×7
 - A_3 of dimensions 7×2 .
- $(A_1 A_2) A_3$ takes $3 \cdot 5 \cdot 7 + 3 \cdot 7 \cdot 2 = 147$
- $A_1 (A_2 A_3)$ takes $5 \cdot 7 \cdot 2 + 3 \cdot 5 \cdot 2 = 100$
- Even though, both calculations return same result

Lecture 5 CS 212 - Arora

Step 1: Notation

17

- Let M_{ij} denote the cost of multiplying $A_i \dots A_j$, (measured in the number of scalar multiplications)
- $M(i, i) = 0$ for all i
- $M(1, n)$ is what we are looking for.

Lecture 5 CS 212 - Arora

Step 2: Proving Principle of Optimality

18

- Every way of multiplying a sequence of matrices can be represented by a binary (infix) tree, where the leaves are the matrices, and the internal nodes are intermediary products.
- Let tree T corresponds to an optimal sequence for $A_i \dots A_j$.
- T has a left subtree L and a right subtree R . L corresponds to multiplying $B = A_i \dots A_k$, and R to multiplying $C = A_{k+1} \dots A_j$, for some integer k ($i \leq k \leq j-1$).
- $\text{Cost}(T) = \text{cost}(L) + \text{Cost}(R) + \text{cost}(BC)$.
- We need to show that if T is an optimal tree, then
 - L is an optimal tree of $A_i \dots A_k$
 - R is an optimal tree for $A_{k+1} \dots A_j$.
- Prove this by contradiction
 - Consider L' strictly better than L , then we can derive T' better than T

Lecture 5 CS 212 - Arora

Step 3: Recurrence Relation

19

- $M_{ij} = \text{cost}(T)$
 - = $\text{cost}(L) + \text{cost}(R) + \text{cost}(BC)$
 - = $M_{ik} + M_{k+1,j} + r_i c_k c_j$
- We can condition over k to find the “best” value of k – the one that minimizes M_{ij}
- $M_{ij} = \min\{M_{ik} + M_{k+1,j} + r_i c_k c_j \mid i \leq k \leq j-1\}$

Lecture 5 CS 212 - Arora

Step 4: Algorithm

20

- Set array $M[i,j]$
 - Only the upper diagonal values will be used
- Initialize $M[i,i] = 0$, for all i
- Initialize $M[i,i+1] = r[i] * c[i] * c[i+1]$, for all i
- Compute for $j = 2$ to $n-i$
 - $M[i,i+j] = \min_k \{M[i,k] + r[i] * c[k] * c[j] + M[k+1,j]\}$
- Time complexity of algorithm is $O(n^3)$, where n is the number of matrices given
- **The algorithm simply finds the sequence in which to multiply the matrices – not to be confused with matrix multiplication algorithm itself.**

Lecture 5 CS 212 - Arora

All Pairs Shortest Path (APSP)

21

- Input: A weighted graph, represented by its weight matrix W .
- Problem: Find the distance between every pair of nodes

Lecture 5 CS 212 - Arora

APSP – Notation

22

- The nodes are numbered $1..n$
- $A^{(k)}(i,j)$ = Length of the shortest path from node i to node j using nodes $1..k$
- $A^{(0)}(i,j) = W[i,j]$

Lecture 5 CS 212 - Arora

APSP – Prove the Principle of Optimality

23

- As discussed earlier, the portion of a shortest path must be a shortest path as well

Lecture 5 CS 212 - Arora

APSP – Recurrence Relation

24

- $A^{(k)}(i,j)$ either uses the node k or does not
- $A^{(k)}(i,j) = \min\{A^{(k-1)}(i,j), A^{(k-1)}(i,k) + A^{(k-1)}(k,j)\}$

Lecture 5 CS 212 - Arora

APSP - Algorithm

25

```

for i=1 to n do
  for j=1 to n do
     $A^{(0)}(i,j) := W[i,j]$ 

for k=1 to n do
  for i=1 to n do
    for j=1 to n do
       $A^{(k)}(i,j) = \min\{A^{(k-1)}(i,j), A^{(k-1)}(i,k) + A^{(k-1)}(k,j)\}$ 

```

Lecture 5 CS 212 - Arora

APSP – Algorithm (Cont.)

26

- Time complexity is $O(n^3)$
- We also observe that once $A^{(k)}$ has been computed, there is no need for $A^{(k-1)}$
- We can save space by not keeping the old values

Lecture 5 CS 212 - Arora

Summary

27

“Dynamic Programming is a methodology of building an optimal solution by first creating subsolutions to subproblems and storing them, potentially losing some space complexity in exchange for much improved time complexity.”

Lecture 5 CS 212 - Arora

Extra Reading

28

- <http://en.wikipedia.org/wiki/Memoization>

Lecture 5 CS 212 - Arora