CS 91 USACO

Bronze Division

Unit 5: Number Algorithms

LECTURE 22: DYNAMIC PROGRAMMING EXAMPLES

DR. ERIC CHOU IEEE SENIOR MEMBER

Objectives

- Dynamic Programming Paradigm
- Review Fibonacci
- Practice: numtri

Dynamic Programming

SECTION 2

Dynamic Programming

- The classic introductory problem in teaching DP is computing the Fibonacci numbers. As a reminder, the Fibonacci numbers are a sequence starting with 1, 1 where each element in the sequence is the sum of the two previous elements:

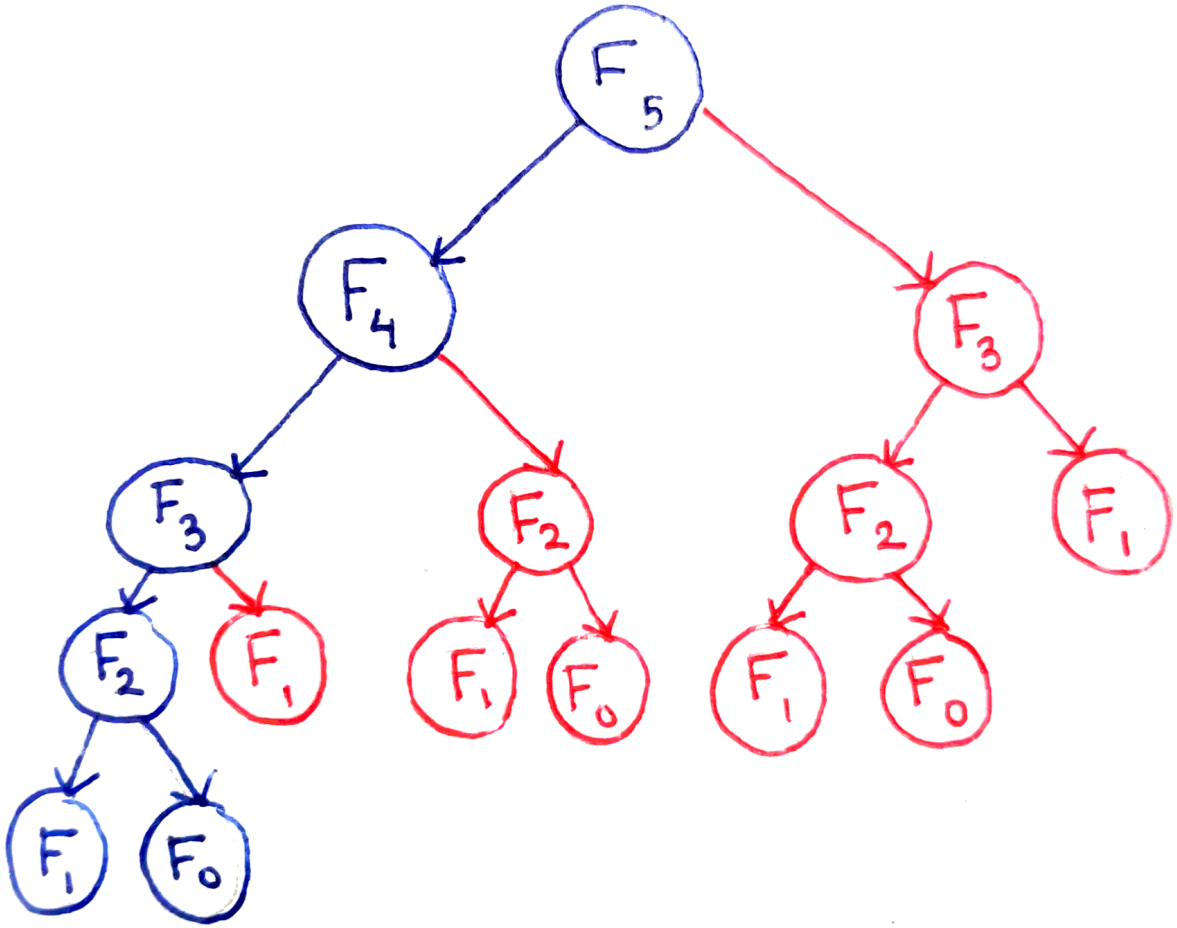
  1, 1, 2, 3, 5, 8, 13, 21,...

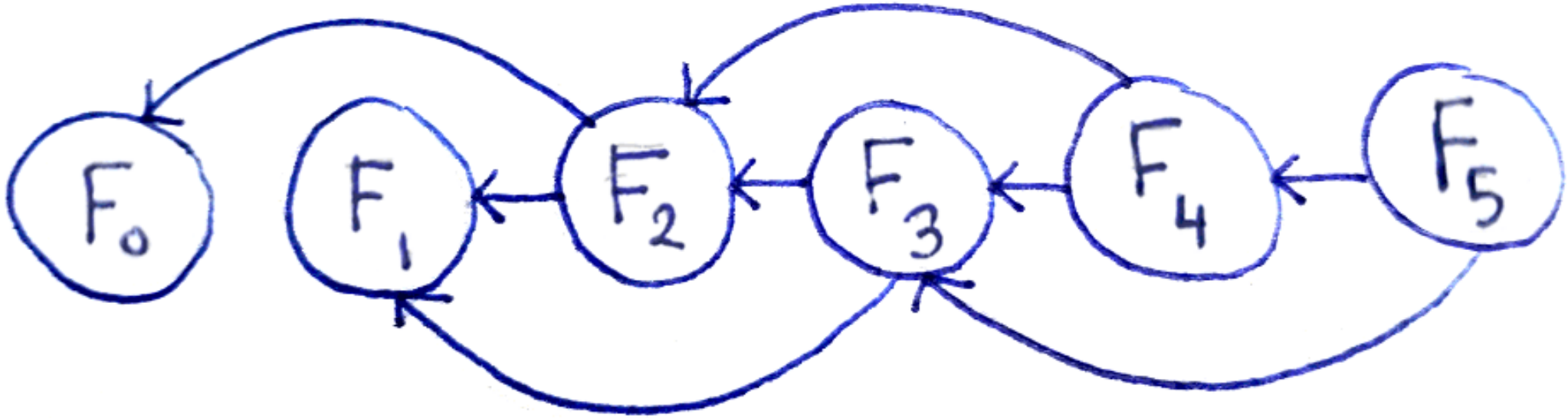
- The canonical recursive formula for the nth element in the sequence is:

$$F_0 = 1$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}$$


F5
F4
F3
F3
F2
F2
F1
F2
F1
F1
F0
F1
F0
F1
F0


F_0
F_1
F_2
F_3
F_4
F_5

Basic Concepts

1. Tabulation vs Memorization
2. Optimal Substructure Property
3. Overlapping Subproblems Property
4. How to solve a Dynamic Programming Problem ?

Tabulation vs Memorization

There are two different ways to store the values so that the values of a sub-problem can be reused. Here, will discuss two patterns of solving dynamic programming (DP) problems:

1. **Tabulation:** Bottom Up
2. **Memoization:** Top Down

Optimal Substructure Property

A given problem is said to have Optimal Substructure Property if the optimal solution of the given problem can be obtained by using the optimal solution to its subproblems instead of trying every possible way to solve the subproblems.

Overlapping Subproblems Property

Dynamic Programming is an algorithmic paradigm that solves a given complex problem by breaking it into subproblems using recursion and storing the results of subproblems to avoid computing the same results again. Following are the two main properties of a problem that suggests that the given problem can be solved using Dynamic programming.

1. Overlapping Subproblems
2. Optimal Substructure

Steps to solve a Dynamic programming problem:

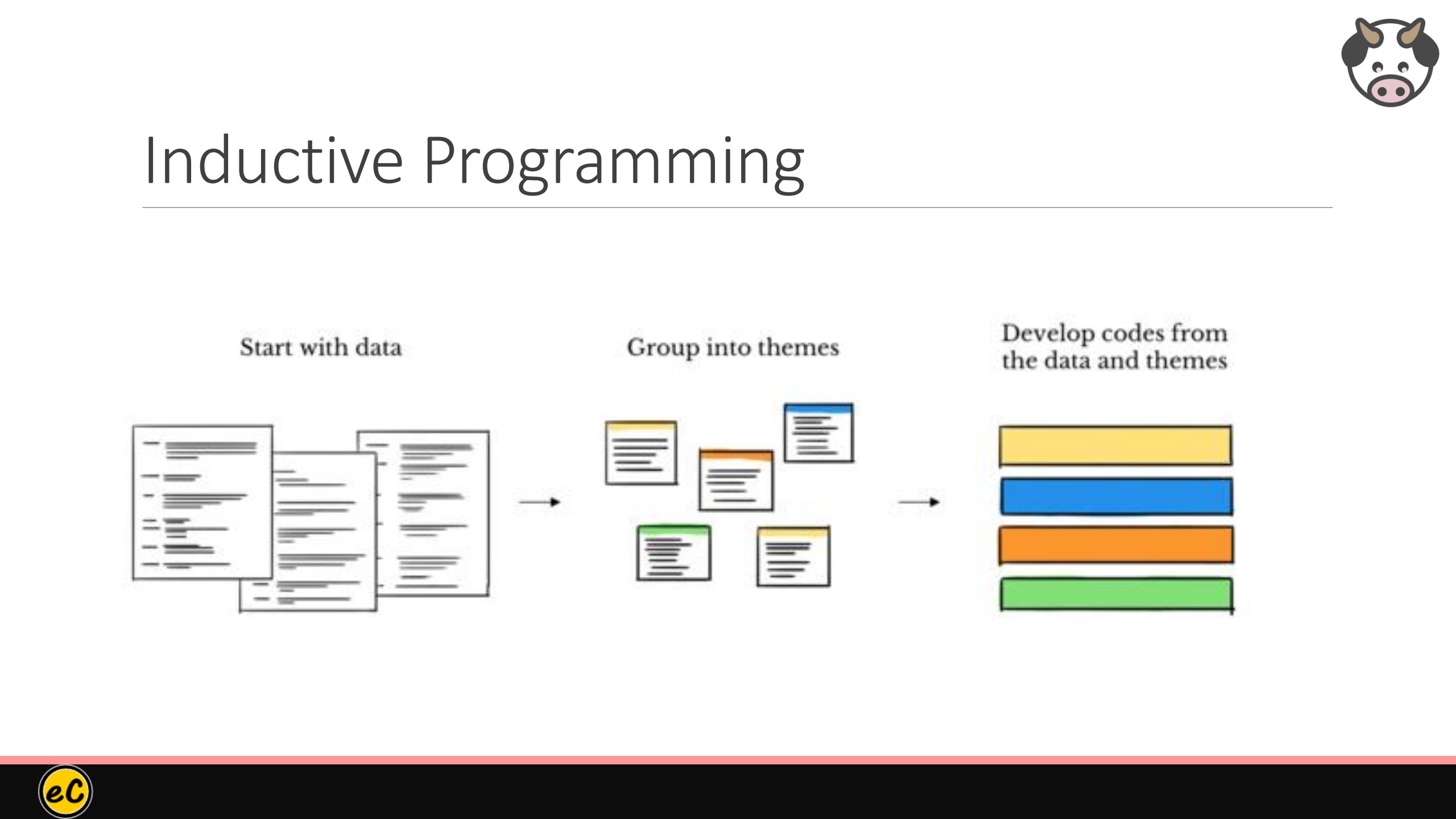
1. Identify if it is a Dynamic programming problem.
2. Decide a state expression with the Least parameters.
3. Formulate state and transition relationship.
4. Do tabulation (or memorization).

Inductive Versus Deductive Programming

SECTION 2

Inductive Programming

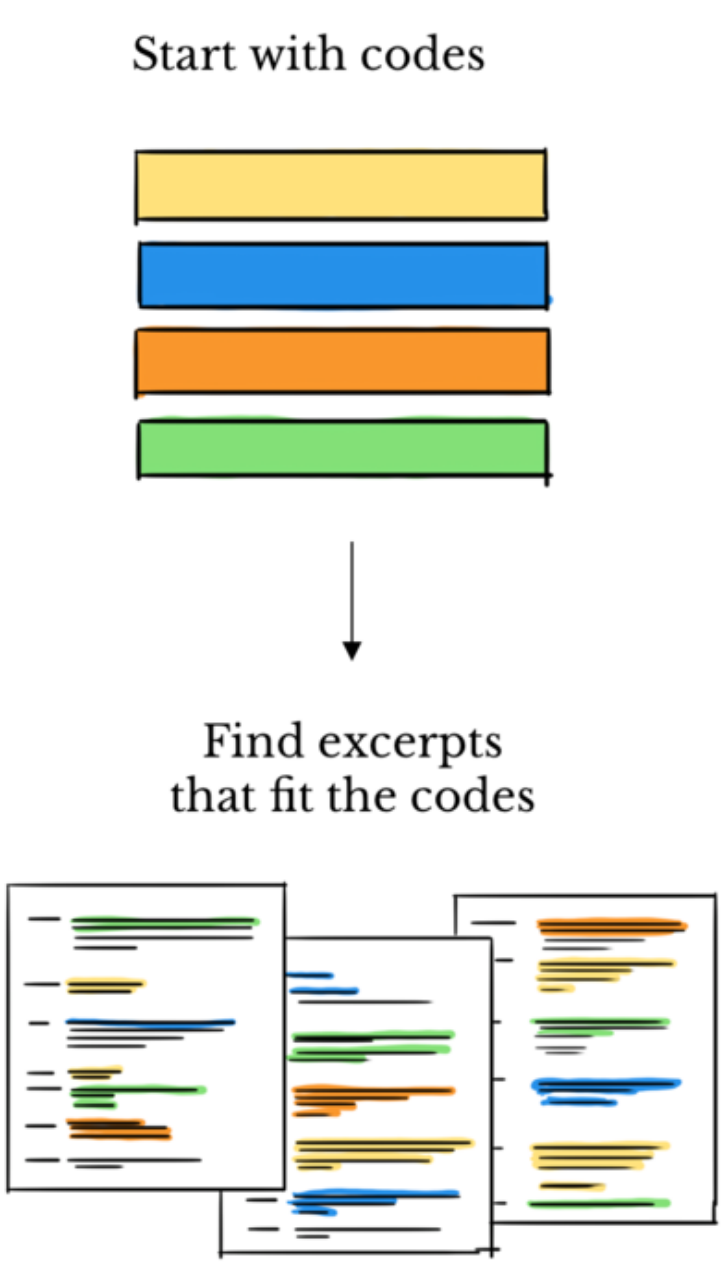
- Inductive programming is the inference of an algorithm or program featuring recursive calls or repetition control structures, starting from information that is known to be incomplete, called the evidence, such as positive and negative input–output examples or clausal constraints. The inferred program must be correct with respect to the provided evidence, in a generalization sense: it should neither be equivalent nor inconsistent to it. Inductive programming is guided explicitly or implicitly by a language bias and a search bias. The inference may draw on background knowledge or query an oracle. In addition to induction, abduction may be used.
- The restriction to algorithms and programs featuring recursive calls or repetition control structures distinguishes inductive programming from concept learning or classification.



Inductive Programming

Deductive Programming

- Deductive coding is a top down approach where you start by developing a codebook with your initial set of codes. This set could be based on your research questions or an existing research framework or theory. You then read through the data and assign excerpts to codes. At the end of your analysis, your codes should still closely resemble the codebook that you started off with. This is good when you have a pre-determined structure for how you need your final findings to be. For example, you may practice deductive data analysis and deductive approaches when doing program evaluation studies or content analysis.



Deductive Programming

Practice: Number Triangle (numtri)

SECTION 4

Problem Statement

- Consider the number triangle shown below. Write a program that calculates the highest sum of numbers that can be passed on a route that starts at the top and ends somewhere on the base. Each step can go either diagonally down to the left or diagonally down to the right.

```
          7
        3   8
      8   1   0
    2   7   4   4
  4   5   2   6   5
```

- In the sample above, the route from 7 to 3 to 8 to 7 to 5 produces the highest sum: 30.

INPUT FORMAT (file numtri.in):

- The first line contains R (1 <= R <= 1000), the number of rows. Each subsequent line contains the integers for that particular row of the triangle. All the supplied integers are non-negative and no larger than 100.

SAMPLE INPUT

```
5
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
```

OUTPUT FORMAT (file numtri.out):

A single line containing the largest sum using the traversal specified.

SAMPLE OUTPUT

30

Nature of Problem

- Simple Dynamic Programming Case (Bottom UP)

INPUT	SUM	SUM	SUM	SUM	SUM
5					
7	0	0	0	0	30
3 8	0 0	0 0	0 0	23 21	23 21
8 1 0	0 0 0	0 0 0	20 13 10	20 13 10	20 13 10
2 7 4 4	0 0 0 0	7 12 10 10	7 12 10 10	7 12 10 10	7 12 10 10
4 5 2 6 5	4 5 2 6 5	4 5 2 6 5	4 5 2 6 5	4 5 2 6 5	4 5 2 6 5

Nature of Problem

- Simple Dynamic Programming Case (Top Down)

INPUT	SUM	SUM	SUM	SUM
5				
7	7			
3 8	10 15	10 15		
8 1 0		18 16 15	18 16 15	
2 7 4 4			20 25 20 19	20 25 20 19
4 5 2 6 5				24 30 27 26 24