Class 20

DYNAMIC PROGRAMMING

Logistics

Please arrive for class ON TIME Monday (and everyday)

Convocation Schedule on Monday for MLK Day

Regular Meeting Time		Meeting time on 1/16
8:00 – 8:50	→	8:00 – 8:40
9:00 – 9:50	→	8:50 – 9:30
10:00 – 10:50	→	9:40 – 10:20
11:00 – 11:50	→	10:30 – 11:10
None	→	Convocation Period
12:00 – 12:50	→	1:00 – 1:40
1:00 – 1:50	→	1:50 – 2:30
2:00 – 2:50	→	2:40 – 3:20
3:00 – 3:50	→	3:30 – 4:10
4:00 – 4:50	→	4:20 – 5:00

Logistics

Exam 2 on Thursday, January 19

- 7 - 8:30pm
- Section 01 - Crapo G219
- Section 02 - Crapo G220
- Covers material from December 12 through January 12: Section 4.5, Chapters 5, 6, and 7
- You may use a 1-page cheat sheet during the exam

No class next Friday, January 20 in exchange for evening exam

Next week is advising week – make sure you discuss your registration plan with them!

Dynamic Programming

Technique for solving problems with overlapping sub-problems.

Sub-problems arise from a recurrence relating a given problem's solution to solutions of its smaller sub-problems.

Rather than solving overlapping sub-problems over and over again, solve each of the smaller sub-problems once and record solutions in a table.

Next time, look up solutions in the table.

Invented by American mathematician Richard Bellman in the 1950s to solve optimization problems

- "Programming" here means "planning," not computer programming

Example: Fibonacci

Problem: compute the *n*th Fibonacci number
Recursive definition: $f_0=0$, $f_1=1$, $f_n = f_{n-1} + f_{n-2}$.
First several: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …

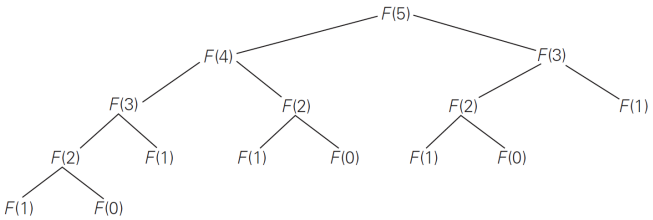


FIGURE 2.6 Tree of recursive calls for computing the 5th Fibonacci number by the definition-based algorithm.

Example: Fibonacci

Problem: compute the *n*th Fibonacci number
Recursive definition: $f_0=0$, $f_1=1$, $f_n = f_{n-1} + f_{n-2}$.
First several: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …

```
private static long fib_array_helper(int n, long[] fibs) {
   if (n == 1) return fibs[1];
   if (n == 2) return fibs[2];
   if (fibs[n] == 0) fibs[n] = fib_array_helper(n-1, fibs) +
                               fib_array_helper(n-2, fibs);
   return fibs[n];
}
```

Running time is Θ(n)

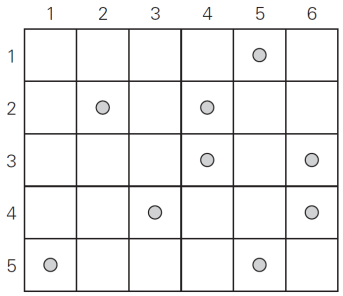
Robot Coin Collection

Coins are placed on an *n* x *m* board.
At most one coin per cell.

Robot:

- starts out in <1,1>
- Deposit at cell <*n,m*>
- On each step, either move:
  - down one cell, or
  - right one cell
- When robot visits a cell with a coin, it automatically gets picked up

Collect maximum # of coins.



Robot Coin Collection

Let *F*(*i, j*) be the largest number of coins the robot can collect and bring to *cell*(*i, j*).

It can reach that cell either from *cell*(*i-1, j*) above or from *cell*(*i, j-1*) to the left.

The largest number of coins that can be brought to those cells are *F*(*i-1, j*) and *F*(*i-1, j*), respectively.

The largest number of coins a robot can bring to *cell*(*i, j*) is the maximum of those two numbers, plus one possible coin at *cell*(*i, j*):

$$F(i, j) = \max\{F(i-1, j), F(i, j-1)\} + c_{ij} \quad \text{for } 1 \le i \le n, \quad 1 \le j \le m$$
$$F(0, j) = 0 \text{ for } 1 \le j \le m \quad \text{and} \quad F(i, 0) = 0 \text{ for } 1 \le i \le n,$$

Robot Coin Collection

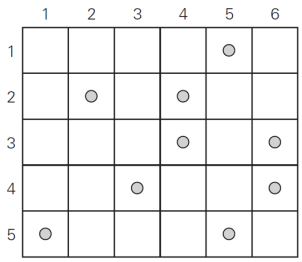
Using the DP formulation, we can fill in an *n* x *m* table for *F*(*i, j*).

ALGORITHM *RobotCoinCollection*$(C[1..n, 1..m])$

```
//Applies dynamic programming to compute the largest number of
//coins a robot can collect on an n × m board by starting at (1, 1)
//and moving right and down from upper left to down right corner
//Input: Matrix C[1..n, 1..m] whose elements are equal to 1 and 0
//for cells with and without a coin, respectively
//Output: Largest number of coins the robot can bring to cell (n, m)
F[1, 1] ← C[1, 1];   for j ← 2 to m do F[1, j] ← F[1, j − 1] + C[1, j]
for i ← 2 to n do
    F[i, 1] ← F[i − 1, 1] + C[i, 1]
    for j ← 2 to m do
        F[i, j] ← max(F[i − 1, j], F[i, j − 1]) + C[i, j]
return F[n, m]
```

Robot Coin Collection

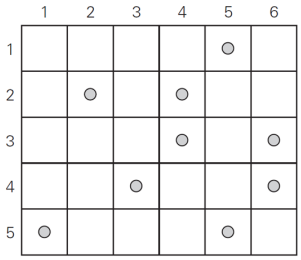
DP results for the problem posted earlier:



	1	2	3	4	5	6
1	0	0	0	0	1	1
2	0	1	1	2	2	2
3	0	1	1	3	3	4
4	0	1	2	3	3	5
5	1	1	2	3	4	**5**

Robot Coin Collection

What if we need the path, in addition to the final coin count?

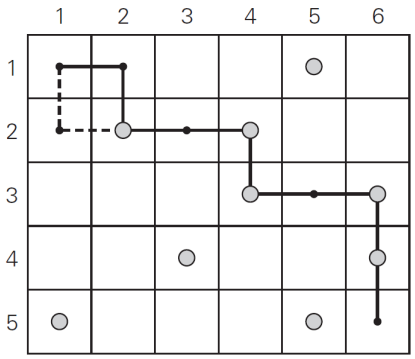


	1	2	3	4	5	6
1	0	0	0	0	1	1
2	0	1	1	2	2	2
3	0	1	1	3	3	4
4	0	1	2	3	3	5
5	1	1	2	3	4	**5**

Robot Coin Collection

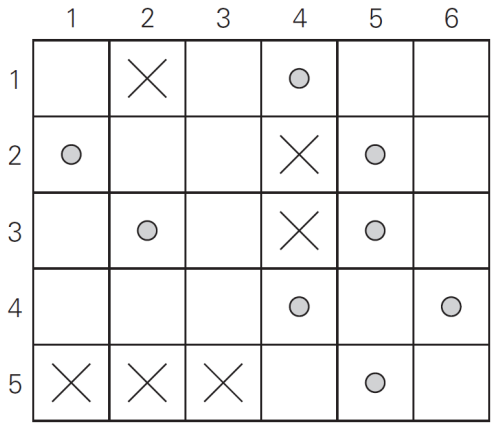
Two optimal paths:

	1	2	3	4	5	6
1	0	0	0	0	1	1
2	0	1	1	2	2	2
3	0	1	1	3	3	4
4	0	1	2	3	3	5
5	1	1	2	3	4	**5**

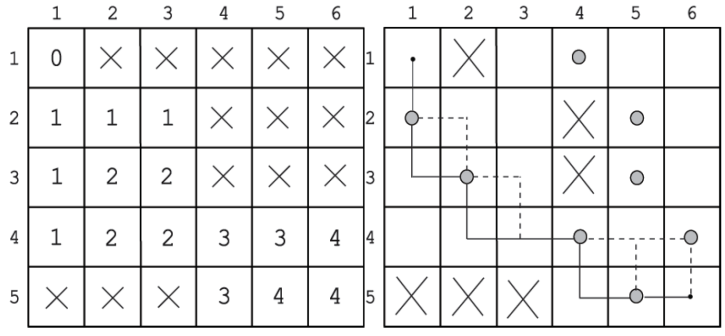


Robot Coin Collection Modified

How do we need to modify our algorithm if some cells are inaccessible to the robot?



Robot Coin Collection Modified



Maximum of 4 coins, 12 possible paths.

Problem Types Where DP Excels

Optimization problems: "minimize …" or "maximize …"

Counting problems: "count the number of ways…"

Often only need the optimal/total value. If also need the solution itself, can *backtrack*

Midterm Reflection for CSSE/MA 473

Please fill in the form distributed in class to hand in.

Honest answers will help you succeed in the course!

Logistics

Please arrive for class ON TIME Monday (and everyday)

Convocation Schedule on Monday for MLK Day

Regular Meeting Time		Meeting time on 1/16
8:00 – 8:50	→	8:00 – 8:40
9:00 – 9:50	→	8:50 – 9:30
10:00 – 10:50	→	9:40 – 10:20
11:00 – 11:50	→	10:30 – 11:10
None	→	Convocation Period
12:00 – 12:50	→	1:00 – 1:40
1:00 – 1:50	→	1:50 – 2:30
2:00 – 2:50	→	2:40 – 3:20
3:00 – 3:50	→	3:30 – 4:10
4:00 – 4:50	→	4:20 – 5:00