Class 22

DYNAMIC PROGRAMMING PRACTICE

DP Review

DP is a technique for solving problem with overlapping sub-problems.

Typically, these sub-problems arise from a **recurrence** relating a given problem's solution to that of its smaller sub-problems.

Additionally, rather than solving sub-problems over and over again, solve once and store in a table.

Dynamic Programming for Knapsack

We get:

$$F(i, j) = \begin{cases} \max\{F(i-1, j),\ v_i + F(i-1, j-w_i)\} & \text{if } j - w_i \geq 0, \\ F(i-1, j) & \text{if } j - w_i < 0. \end{cases}$$

It is convenient to define the initial conditions as:

$$F(0, j) = 0 \text{ for } j \geq 0 \quad \text{and} \quad F(i, 0) = 0 \text{ for } i \geq 0.$$

DP for Knapsack Analysis

Table has nW entries

To determine each entry, 2 options. 1 subproblem for each.

Efficiency: $\Theta(nW)$

Can improve slightly if we work top-down recursively, and only compute table values once (we say the function computations are memoized). Will only compute table entries necessary to solve the overall problem

- Faster, but in most cases, stays $\Theta(nW)$

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*):

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

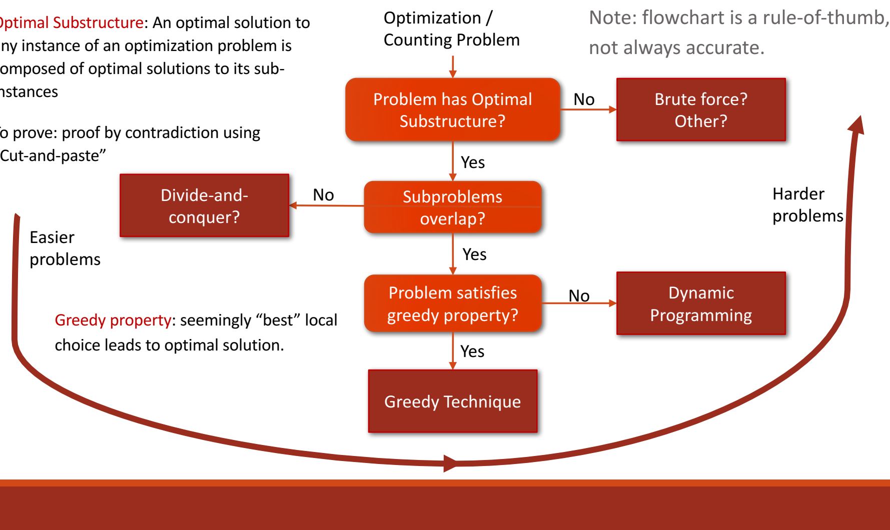
Example: Fibonacci

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)

When to Use DP?



DP Technique

Parameterize solutions to subproblems

Derive recurrence relating solution to problem to solution of subproblems

Would it benefit from top-down memoization?
- If not, use standard bottom-up DP.

Think about DP table
- dimensions, size, how filled

Analysis: size of table, how much work per entry?

Dynamic Programming Examples

"Three basic examples" from Sec. 8.1

Some difficult discrete optimization problems:
- knapsack (Sec. 8.2)
- traveling salesman

Constructing an optimal binary search tree (Sec. 8.3)

Warshall's algorithm for transitive closure (Sec. 8.4)

Floyd's algorithm for all-pairs shortest paths (Sec. 8.4)

Dynamic Programming Examples: Coin-row problem

Complete problem 1 on worksheet

Coin-Row Problem

Recurrence: $F(n) = \max\{c_n + F(n-2),\ F(n-1)\}$ for $n > 1$,

$F(0) = 0, \quad F(1) = c_1.$

Algorithm:

ALGORITHM *CoinRow*($C[1..n]$)
//Applies formula (8.3) bottom up to find the maximum amount of money
//that can be picked up from a coin row without picking two adjacent coins
//Input: Array $C[1..n]$ of positive integers indicating the coin values
//Output: The maximum amount of money that can be picked up
$F[0] \leftarrow 0$; $F[1] \leftarrow C[1]$
for $i \leftarrow 2$ **to** n **do**
  $F[i] \leftarrow \max(C[i] + F[i-2],\ F[i-1])$
return $F[n]$

Coin-Row Sample Run

Runtime: $\Theta(n)$

Space: $\Theta(n)$

$F[0] = 0$, $F[1] = c_1 = 5$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5					

$F[2] = \max\{1 + 0, 5\} = 5$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5				

$F[3] = \max\{2 + 5, 5\} = 7$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7			

$F[4] = \max\{10 + 5, 7\} = 15$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15		

$F[5] = \max\{6 + 7, 15\} = 15$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15	15	

$F[6] = \max\{2 + 15, 15\} = 17$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15	15	**17**

Dynamic Programming Examples

Complete problem 2 on worksheet