Dynamic Programming

We'd like to have "generic" algorithmic paradigms for solving problems

Example: Divide and conquer

- Break problem into independent subproblems
- Recursively solve subproblems (subproblems are smaller instances of main problem)
- Combine solutions

Examples:

- Mergesort,
- Quicksort,
- Strassen's algorithm
- ...

Dynamic Programming: Appropriate when you have recursive subproblems that are not independent

Example: Making Change

Problem: A country has coins with denominations

$$1 = d_1 < d_2 < \cdots < d_k.$$

You want to make change for n cents, using the smallest number of coins.

Example: U.S. coins

$$d_1 = 1 \quad d_2 = 5 \quad d_3 = 10 \quad d_4 = 25$$

Change for **37 cents** − 1 quarter, 1 dime, 2 pennies.

What is the algorithm?

Change in another system

Suppose

$$d_1 = 1 \quad d_2 = 4 \quad d_3 = 5 \quad d_4 = 10$$

- Change for 7 cents – 5,1,1
- Change for 8 cents – 4,4

What can we do?

Change in another system

Suppose

$$d_1 = 1 \quad d_2 = 4 \quad d_3 = 5 \quad d_4 = 10$$

- Change for 7 cents – 5,1,1
- Change for 8 cents – 4,4

What can we do?

The answer is counterintuitive. To make change for n cents, we are going to figure out how to make change for every value $x < n$ first. We then build up the solution out of the solution for smaller values.

Solution

We will only concentrate on computing the number of coins. We will later recreate the solution.

- Let $C[p]$ be the minimum number of coins needed to make change for p cents.

- Let x be the value of the first coin used in the optimal solution.

- Then $C[p] = 1 + C[p - x]$.

Problem: We don't know x.

Solution

We will only concentrate on computing the number of coins. We will later recreate the solution.

- Let $C[p]$ be the minimum number of coins needed to make change for p cents.

- Let x be the value of the first coin used in the optimal solution.

- Then $C[p] = 1 + C[p - x]$.

Problem: We don't know x.

Answer: We will try all possible x and take the minimum.

$$C[p] = \begin{cases} \min_{i:d_i \leq p}\{C[p - d_i] + 1\} & \textbf{if } p > 0 \\ 0 & \textbf{if } p = 0 \end{cases}$$

Example: penny, nickel, dime

$$C[p] = \begin{cases} \min_{i:d_i \le p}\{C[p-d_i]+1\} & \textbf{if } p > 0 \\ 0 & \textbf{if } p = 0 \end{cases}$$

CHANGE(**p**)

1 **if** $(p < 0)$
2   **then return** ∞
3 **elseif** $(p = 0)$
4   **then return 0**
5   **else**
6 **return** $1 + \min\{$CHANGE$(p-1),$ CHANGE$(p-5),$ CHANGE$(p-10)\}$

What is the running time?   (don't do analysis here)

Dynamic Programming Algorithm

DP-CHANGE(**n**)

1   $C[< 0] = \infty$

2   $C[0] = 0$

3   **for** $p = 2$ **to** n

4   **do** $min = \infty$

5   **for** $i = 1$ **to** k

6   **do if** $(p \geq d_i)$

7   **then if** $(C[p - d_i]) + 1 < min)$

8   **then** $min = C[p - d_i] + 1$

9                          $coin = i$

10

11          $C[p] = min$

12          $S[p] = coin$

Running Time: $O(nk)$

Dynamic Programming

Used when:

- Optimal substructure - the optimal solution to your problem is composed of optimal solutions to subproblems (each of which is a smaller instance of the original problem)

- Overlapping subproblems

Methodology

- Characterize structure of optimal solution

- Recursively define value of optimal solution

- Compute in a bottom-up manner

Example: Rod Cutting

Problem: Given a rod of length n inches and a table of prices p_i for $i = 1, 2, \ldots, n$, determine the maximum revenue r_n obtainable by cutting up the rod and selling the pieces.

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

How can we cut a rod of length 4?

Optimal Substructure

Suppose that we know that optimal solution makes the first cut to be length k, then the optimal solution consists of an optimal solution to the remaining piece of length $n - k$, plus the first piece of length k

Suppose not. Then we are saying that the optimal solution consists of some way to cut the piece of length $n - k$ that is not optimal, plus the piece of length k. Let p_k be the profit from the piece of length k, and let y be profit from the non-optimal solution to the piece of length $n - k$. Then we are receiving a total profit of $y + p_k$. Now suppose that instead of the proposed solution to the piece of length k, we used an optimal solution to the piece of length k instead. Let y' be the profit associated with the optimal solution to the piece of length $n - k$, and since it is optimal $y' > y$. We could then put this together with the piece of length k and obtain a solution of profit $y' + k > y + k$, contradicting the claim that the original solution was optimal.

Recursive Implementation

Recurrence

$$r_n = \max_{1 \le i \le n} (p_i + r_{n-i}) \ . \tag{1}$$

Code

$Cut - Rod(p, n)$

```
1   if  n == 0
2        then return 0
3   q ← −∞
4   for  i ← 1 to  n
5            do q ← max(q, p[i] + Cut-Rod(p, n − i))
6   return  q
```

What is the running time?

DP solution

$Bottom - Up - Cut - Rod(p, n)$

1   **let** $r[0 \mathinner{\ldotp\ldotp} n]$ **be a new array**

2 $r[0] \leftarrow 0$

3   **for** $j \leftarrow 1$ **to** n

4   **do** $q \leftarrow -\infty$

5   **for** $i \leftarrow 1$ **to** j

6   **do** $q \leftarrow \max(q, p[i] + r[j - i])$

7 $r[j] \leftarrow q$

8   **return** $r[n]$

What is the running time?