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

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

Change for 37 cents – 1 quarter, 1 dime, 2 pennies.

What is the algorithm?

Change in another system

Suppose

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

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

What can we do?

Change in another system

Suppose

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

- **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.

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

Example: penny, nickel, dime

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

$\textsc{Change}(\mathbf{p})$

if $(p < 0)$

  return ∞

elseif $(p = 0)$

  return 0

else

return $1 + \min\{\textsc{Change}(p-1), \textsc{Change}(p-5), \textsc{Change}(p-10)\}$

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

Dynamic Programming Algorithm

```
DP-CHANGE(n)
C[< 0] = ∞
C[0] = 0
for p = 2 to n
    min = ∞
    for i = 1 to k
        if (p ≥ d_i)
            if (C[p − d_i]) + 1 < min)
                min = C[p − d_i] + 1
                coin = i

    C[p] = min
    S[p] = coin
```

Running Time: $O(nk)$

Dynamic Programming

Used when:

- **Optimal substructure**
- **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

Proof. **Suppose not. Then we are saying that the optimal solution to the whole problem with a first cut at k, consists of a non-optimal way to cut the piece of length $n - k$. Let the optimal solution have value X and define $Y = X - p_k$, be the value for the optimal solution to the whole problem associated with the piece of length $n - k$. Since we are cutting the piece of length $n - k$ non-optimally, then we must have that we could have cut and received Y' profit, where $Y' > Y$. But then we can use this scheme to get a solution for the whole problem of value**

$$p_k + Y' > p_k + Y = X,$$

which contradicts the optimality of our original solution.

Recursive Implementation

Recurrence

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

Code

$Cut - Rod(p, n)$

```
if n == 0
        return 0
q = -∞
for i = 1 to n
        q = max(q, p[i] + Cut-Rod(p, n - i))
return q
```

What is the running time?

DP solution

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

let $r[0 \,..\, n]$ **be a new array**
$r[0] = 0$
for $j = 1$ **to** n
    $q = -\infty$
    for $i = 1$ **to** j
        $q = \max(q, p[i] + r[j - i])$
    $r[j] = q$
return $r[n]$

What is the running time?