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.

$$C[p] = \begin{cases} \min_{i:d_i \leq p}\{C[p - d_i] + 1\} & \text{if } p > 0 \\ 0 & \text{if } p = 0 \end{cases}$$
Example: penny, nickel, dime

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

\text{Change}(p)
1 \text{ if } (p < 0)
2 \quad \text{then return } \infty
3 \text{ elseif } (p = 0)
4 \quad \text{then return } 0
5 \text{ else}
6 \text{ return } 1 + \min \{ \text{Change}(p - 1), \text{Change}(p - 5), \text{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. $coin = i$
4. $C[p] = min$
5. $S[p] = coin$

**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