# LECTURE 7

## DYNAMIC PROGRAMMING

CS200

March 18, 2022

BOSTON UNIVERSITY

# Minimizing Coins (CSES)

Given infinite coins of `n` different denominations, find the minimum number of coins needed to produce a value `x`. Print `-1` if it is not possible.

- Example 1: `n` $= 3$ denominations: `[1, 5, 7]`, `x` $= 11$
- Output: `3`
- Explanation: 3 coins minimum: $5 + 5 + 1 = 11$

- Example 2: `n` $= 2$ denominations: `[4, 9]`, `x` $= 3$
- Output: `-1`
- Explanation: impossible

# DP-Recurrence

Observation: $dp[i] = min(dp[i - C_1] + 1, dp[i - C_2] + 1, ..., dp[i - C_n] + 1)$
for i = 1...x assuming $dp[i - C_j]$ != $-1$

# Dynamic Programming

Algorithmic technique that helps us break down hard problems

- Most of the time, exponential $\longrightarrow$ pseudo-polynomial
- We will assume we can compute in polynomial time.

Two main uses

- Finding optimal solution (large as possible/small as possible)
- Finding the number of solutions

Find sub-problems and leverage memoization to avoid re-solving!

# Edit Distance

Problem: Given two strings word1 and word2, return the minimum number of operations required to convert word1 to word2.

You have the following three operations permitted on a word:

- Insert a character
- Delete a character
- Replace a character

Let's do an example with word1 = intention and `word2 = execution`

- intention $\rightarrow$ inention (remove 't')
- inention $\rightarrow$ enention (replace 'i' with 'e')
- enention $\rightarrow$ exention (replace 'n' with 'x')
- exention $\rightarrow$ exection (replace 'n' with 'c')
- exection $\rightarrow$ execution (insert 'u')

# Edit Distance Strategy

Cannot brute force.

If `word1[i - 1] == word2[j - 1]`, we do not need to make any change. Therefore `dp[i][j] = dp[i - 1][j - 1]`

If `word1[i - 1] != word2[j - 1]`, we need to consider three cases.

- Replace word1[i - 1] by word2[j - 1] (dp[i][j] = dp[i - 1][j - 1] + 1)
- If word1[0..i - 1) = word2[0..j) then delete word1[i - 1] (dp[i][j] = dp[i - 1][j] + 1)
- If word1[0..i) + word2[j - 1] = word2[0..j) then insert word2[j - 1] to word1[0..i) (dp[i][j] = dp[i][j - 1] + 1)