The Coin Changing problem

- Suppose we need to make change for 67¢. We want to do this using the fewest number of coins possible. Pennies, nickels, dimes and quarters are available.
- Optimal solution for 67¢ has six coins: two quarters, one dime, a nickel, and two pennies.
- We can use a **greedy algorithm** to solve this problem: repeatedly choose the largest coin less than or equal to the remaining sum, until the desired sum is obtained.
- This is how millions of people make change every day (*).

CS404/504

The Coin-Changing problem: formal description

- Let D = {d₁, d₂, ..., d_k} be a finite set of distinct coin denominations. Example: d₁ = 25¢, d₂ = 10¢, d₃ = 5¢, and d₄ = 1¢.
- We can assume each d_i is an integer and $d_1 > d_2 > ... > d_k$.
- Each denomination is available in unlimited quantity.
- The Coin-Changing problem:
 - Make change for n cents, using a minimum total number of coins.
 - Assume that $d_k = 1$ so that there is always a solution.

CS404/504

The Greedy Method (works in the US) • For the coin set { 25c, 10c, 5c, 1c}, the greedy method always finds the optimal solution. • **Exercise:** prove it. • It may not work for other coin sets. For example it stops working if we knock out the nickel. • Example: $D = \{ 25c, 10c, 1c \}$ and n = 30c. The Greedy method would produce a solution: $25c + 5 \times 1c$, which is not as good as $3 \times 10c$.

CS404/504

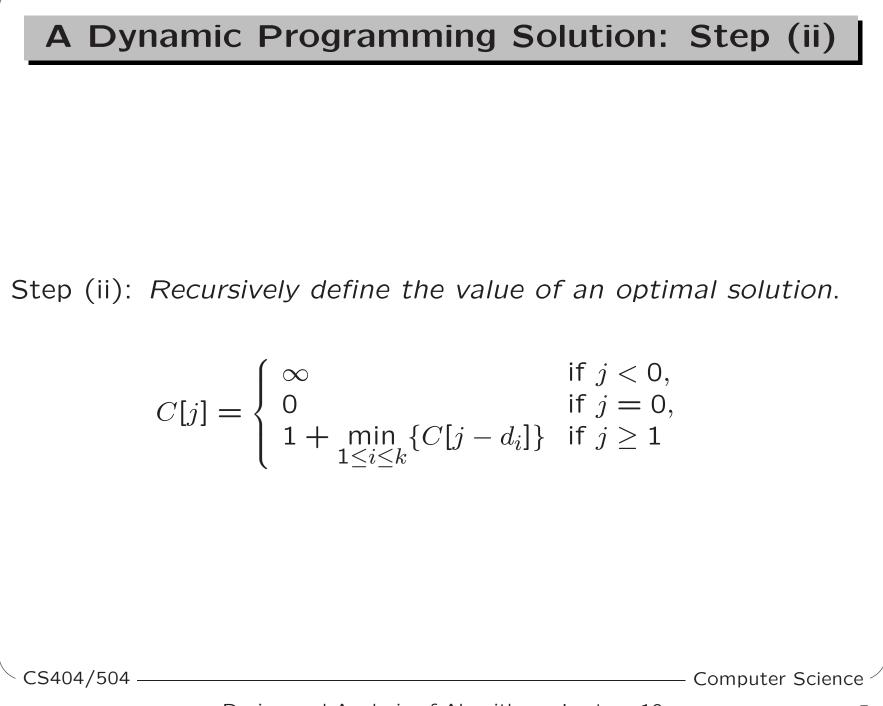
A Dynamic Programming Solution: Step (i)

Step (i): Characterize the structure of a coin-change solution.

- Define C[j] to be the minimum number of coins we need to make change for j cents.
- If we knew that an optimal solution for the problem of making change for j cents used a coin of denomination d_i, we would have:

 $C[j] = 1 + C[j - d_i].$

CS404/504



An example: coin set { 50¢, 25¢, 10¢, 1¢}

C[0] = 0;

CS404/504

$$C[1] = \min \begin{cases} 1 + C[1 - 50] = \infty \\ 1 + C[1 - 25] = \infty \\ 1 + C[1 - 10] = \infty \\ 1 + C[1 - 1] = 1 \end{cases}$$
$$C[2] = \min \begin{cases} 1 + C[2 - 50] = \infty \\ 1 + C[2 - 25] = \infty \\ 1 + C[2 - 10] = \infty \\ 1 + C[2 - 1] = 2 \end{cases}$$

Similarly, C[3] = 3; C[4] = 4; ...; C[9] = 9; C[10] = 1;

- Computer Science

An example

$$C[11] = \min \begin{cases} 1 + C[11 - 50] = \infty \\ 1 + C[11 - 25] = \infty \\ 1 + C[11 - 10] = 2 & \{ 1^{\circ}, 10^{\circ} \} \\ 1 + C[11 - 1] &= 2 & \{ 10^{\circ}, 1^{\circ} \} \end{cases}$$

$$C[20] = 2; \dots, C[29] = 5;$$

$$C[30] = \min \begin{cases} 1 + C[30 - 50] = \infty \\ 1 + C[30 - 25] = 1 + C[5] = 6 \\ 1 + C[30 - 10] &= 1 + C[20] = 3; \\ 1 + C[30 - 1] &= 1 + C[29] = 6; \end{cases}$$

$$C(5404/504 \qquad \qquad Computer Science$$

A Dynamic Programming Solution: Step (iii)

Step (iii): Compute values in a bottom-up fashion.

```
Avoid examining C[j] for j < 0 by ensuring that j \ge d_i before looking up C[j - d_i].
```

```
COMPUTE-CHANGE(n, d, k)
   C[0] := 0
   for j := 1 to n do
     C[j] := \infty
      for i := 1 to k do
        if j \geq d_i and 1 + C[j - d_i] < C[j] then
           C[j] := 1 + C[j - d_i]
   return c
Complexity: \Theta(nk).
CS404/504 -
                                                       Computer Science
```

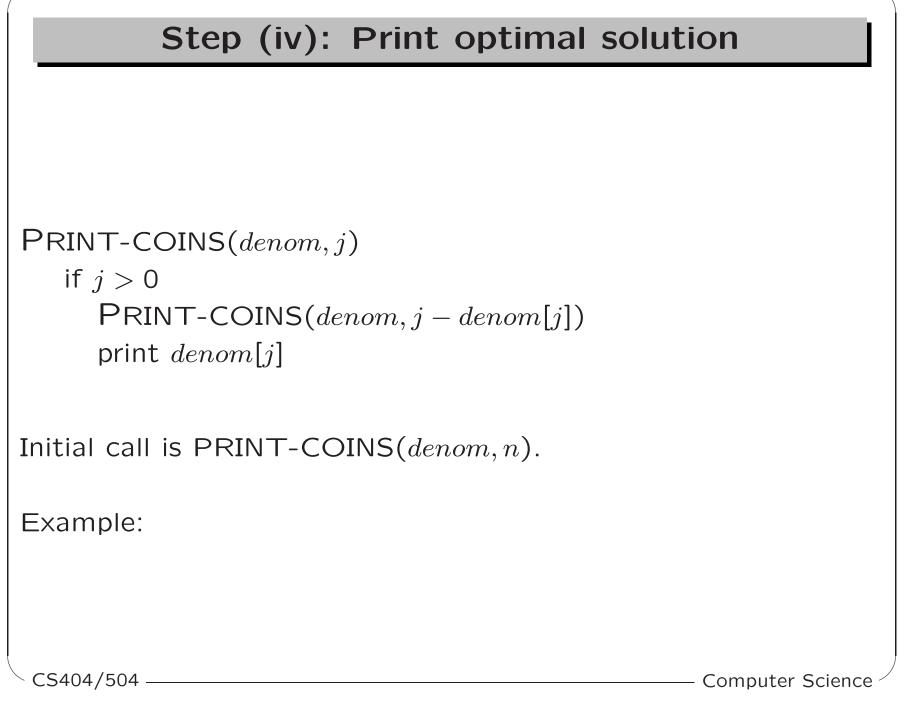
A Dynamic Programming Solution: Step (iv)

Step (iv): Construct an optimal solution.

We use an additional array denom[1..n], where denom[j] is the denomination of a coin used in an optimal solution to the problem of making change for j cents.

```
COMPUTE-CHANGE(n, d, k)
```

```
C[0] := 0
for j := 1 to n do
C[j] := \infty
for i := 1 to k do
if j \ge d_i and 1 + C[j - d_i] < C[j] then
C[j] := 1 + C[j - d_i]
denom[j] := d_i
return c
```



Time complexity of DP algorithms

Usually the complexity of a DP algorithm is: # of sub-problems × choices for each sub-problem

For example: **0/1 Knapsack Problem:** $C[i, \varpi] = \max(C[i-1, \varpi], C[i-1, \varpi - w_i] + p_i).$ $n \times M$ sub-problems, each needs to check **2** choices. $- \Theta(nM)$

Matrix Chain Multiplication:

$$\begin{split} C[i,j] &= \min_{i \leq k < j} \{ C[i,k] + C[k+1,j] + rows[A_i] * col[A_k] * col[A_j] \} \\ \mathbf{n} \times \mathbf{n} \text{ sub-problems, each needs to check } O(n) \text{ choices} \\ &= O(n^3) \end{split}$$

Coin Changing Problem: size of C = n, k possible coin types for each C[j]. — $\Theta(nk)$.

CS404/504

- Let $D = \{d_1, d_2, ..., d_k\}$ be the set of coin denominations, arranged such that $d_1 = 1$ ¢. As before, the problem is to make change for n cents using the fewest number of coins.
- Use a table *C*[1..*k*, 0..*n*]:
 - C[i, j] is the smallest number of coins used to make change for j cents, using only coins $d_1, d_2, ..., d_i$.
 - The overal number of coins (the final optimal solution) will be computed in C[k, n].

CS404/504

Step (i): Characterize the structure of a coin-change solution.

• Making change for *j* cents with coins $d_1, d_2, ..., d_i$ can be done in two ways:

1) Don't use coin d_i (even if it's possible):

C[i,j] = C[i-1,j]

2) Use coin d_i and reduce the amount:

 $C[i, j] = 1 + C[i, j - d_i].$

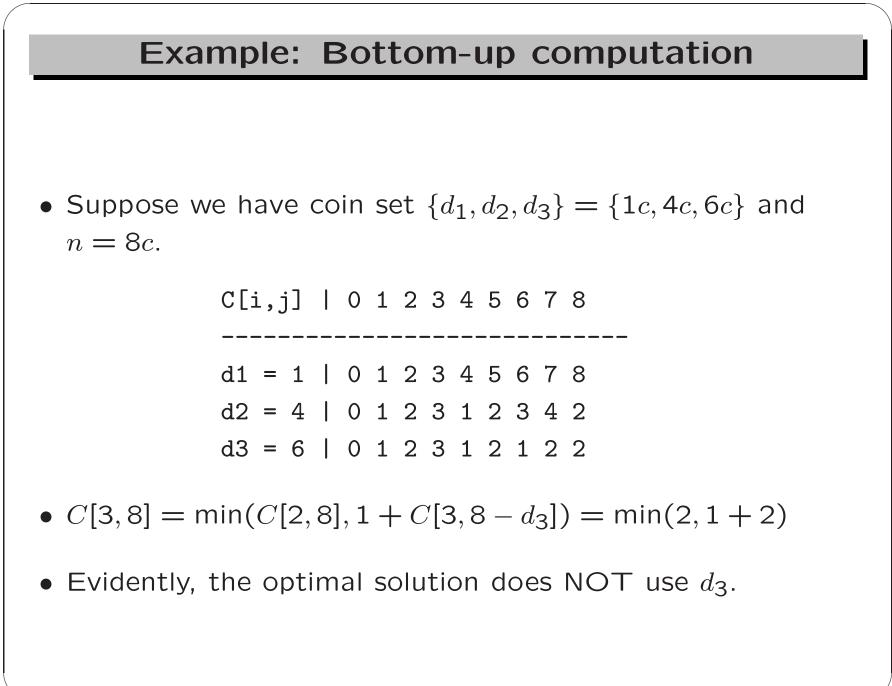
• We will pick the solution with least number of coins:

$$C[i, j] = \min(C[i-1, j], 1 + C[i, j - d_i])$$

CS404/504 -

Another Dynamic Programming Solution Step (ii): Recursively define the value of an optimal solution. $C[i,j] = \begin{cases} \infty & \text{if } j = 0, \\ j & \text{if } i = 0, \\ \min\{C[i-1,j], 1 + C[i,j-d_i]\} & \text{if } j \ge 1 \end{cases}$ CS404/504 **Computer Science**

```
Step (iii): Compute values in a bottom-up fashion.
COMPUTE-CHANGE(d, k, n)
   for i := 1 to k
     C[i, 0] := 0
   for j := 1 to n
     C[1, j] := j
   for i := 1 to k
                               Overall time complexity is \Theta(nk)
      for j := 1 to n
        if j < d_i then
           C[i, j] := C[i - 1, j]
        else
           C[i, j] := \min(C[i-1, j], 1 + C[i, j-d_i])
 CS404/504
                                                      Computer Science
```



CS404/504

Step (iv): Construct an optimal solution.

Two strategies:

- Online: use an additional matrix S[1..k, 0..n], where S[i, j] indicates which of the values C[i 1, j] and C[i, j d_i] was used to compute C[i, j] (use two symbols: ↑ and ←). Compute S in parallel with C.
- Batch: recover the denominations of the coins used in the optimal solution by starting backwards from C[k, n], after computing the entire matrix C.

HW exercise: write the pseudocode for each, analyze time & space complexity.

CS404/504