#### Coin Cashier Problem

Dynamic Programming Approach

#### Dynamic Programming: Coin Cashier Problem

Def. OPT(i,v): min # coins in the subset of coins of index 1,2,...,i to reach value v. If not possible, it is infinity.

Goal. OPT(n, V).

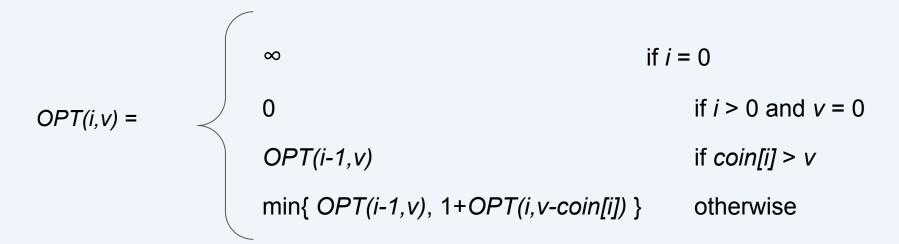
Case 1. OPT(i, v) does not select coin of index i.

• OPT(i,v) selects the best of coins of index {1,2,...,i-1} to reach value v.

Case 2. *OPT(i, v)* selects coin of index *i*.

- # coins used + 1
- New value to reach v-coins[i]
- OPT(i,v) selects the best of coins of index {1,2,...,i} to reach the new value.
   Note that coin of index i can be used again, differently of the knapsack problem.

## Equation



### Algorithm

```
1 V <- Final value to be reached by summing coin values
 2 n <- number of coin types
 3 coin[] <- array of coin values sorted ascending</pre>
 4 M[i,v] <- array of stored subproblem OPT(i,v) solution
 6 \cdot \text{for } V = 0 \text{ to } V
        M[0,v] \leftarrow infinity
 9 \cdot \text{for } i = 1 \text{ to } n
        M[i,0] < -0
10
11
12 \cdot \text{for } V = 1 \text{ to } V
13 -
       for i = 1 to n
14 -
             if coin[i] > v
15
                  M[i,v] = M[i-1,v]
16 -
           else
                  M[i,v] = min(M[i-1,v], 1+M[i,v-coin[i]])
17
18
    return M[n,V]
```

### Coin Cashier Problem: running time

Theorem. The DP algorithm solves the coin cashier problem with n coin types and maximum value V in  $\Theta(n \ V)$  time and  $\Theta(n \ V)$  space.

#### Pf.

- Takes O(1) time per table entry.
- There are  $\Theta(n \ V)$  table entries.
- After computing optimal values, can trace back to find solution:

OPT(i, v) takes item i iff M[i, v] > M[i - 1, v]. If not true, look for immediate above value M[i - 1, v] compared to M[i - 2, v]. When true, look for value v-coin[i], until you get value 0.

# Example: coins [7,8,9] to reach value 15

v ->	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
[]	inf	inf	inf	inf	inf	inf	inf	inf	inf							
[7]	• 0	inf	inf	inf	inf	inf	inf	1	inf	inf	inf	inf	inf	inf	2	inf
[7,8]	0	inf	inf	inf	inf	inf	inf	<del>4 1</del>	1	inf	inf	inf	inf	inf	2	2
[7,8,9]	0	inf	inf	inf	inf	inf	inf	1	1	1	inf	inf	inf	inf	2	2

Solution: coins 8 + 7