#### Fibonacci number

# Coin change problem

Given the denominations  $\mathbf{c}$ :  $c_1$ ,  $c_2$ , ...,  $\underline{c_d}$ , the recurrence relation is:

```
RecursiveChange(M, c, d)
1.
          if M = 0
2.
            return 0
3.
          bestNumCoins ← infinity
4.
          for i \leftarrow 1 to d
5.
            if M \ge \underline{c}_i
6.
               numCoins \leftarrow RecursiveChange(M - c_i, c, d)
7.
8.
                if <u>numCoins</u> + 1 < <u>bestNumCoins</u>
                  bestNumCoins ← numCoins + 1
9.
            return bestNumCoins
10.
     DPChange(M, c, d)
1.
                                                   Running time: O(M^*d)
2.
        bestNumCoins_0 \leftarrow 0
3.
        for m \leftarrow 1 to M
           bestNumCoins_m \leftarrow infinity
4.
5.
           for i \leftarrow 1 to d
6.
             if m \ge \underline{c}_i
                \textbf{if } \underline{bestNumCoins}_{\underline{m}-\underline{c_i}} + 1 < \underline{bestNumCoins}_{\underline{m}}
7.
                   \underline{bestNumCoins_{\underline{m}}} \leftarrow \underline{bestNumCoins_{\underline{m}-c_{\underline{i}}}} + 1
8.
9.
        return bestNumCoins<sub>M</sub>
```

#### Rod cutting

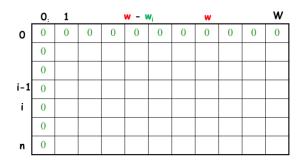
The **rod-cutting problem** is the following. Given a rod of length n inches and a table of prices  $p_i$  for i = 1, 2, ..., n, determine the maximum revenue  $r_n$  obtainable by cutting up the rod and selling the pieces. Note that if the price  $p_n$  for a rod of length n is large enough, an optimal solution may require no cutting at all.

Consider the case when n=4. Figure 15.2 shows all the ways to cut up a rod of 4 inches in length, including the way with no cuts at all. We see that cutting a 4-inch rod into two 2-inch pieces produces revenue  $p_2 + p_2 = 5 + 5 = 10$ , which is optimal.

We can cut up a rod of length n in  $2^{n-1}$  different ways, since we have an independent option of cutting, or not cutting, at distance i inches from the left end,

### 0-1 Knapsack

$$P[i, w] = \begin{cases} P[i-1, w] & \text{if } w < w_i \\ \max\{v_i + P[i-1, w - w_i], P[i-1, w]\} & \text{else} \end{cases}$$



for 
$$\mathbf{w} = 0$$
 to  $\mathbf{W}$ 

$$P[0, \mathbf{w}] = 0$$
for  $\mathbf{i} = 0$  to  $\mathbf{n}$ 

$$P[i, 0] = 0$$
for  $\mathbf{w} = 0$  to  $\mathbf{W}$ 
if  $\mathbf{w}_i <= \mathbf{w}$  // item  $\mathbf{i}$  can be part of the solution
if  $(\mathbf{v}_i + \mathbf{P}[i-1, \mathbf{w}-\mathbf{w}_i] > \mathbf{P}[i-1, \mathbf{w}])$ 

$$P[i, \mathbf{w}] = \mathbf{v}_i + \mathbf{P}[i-1, \mathbf{w}-\mathbf{w}_i]$$
else
$$P[i, \mathbf{w}] = P[i-1, \mathbf{w}]$$
else  $P[i, \mathbf{w}] = P[i-1, \mathbf{w}]$ 

## Subset sum problem

You are given an array A and a number N. You need to find out if N is a sum of any subset of A or not.

Example:

$$A = \{2, 4, 5, 6, 8\}, N = 15 \rightarrow True \{4, 5, 6\}$$

$$A = \{2, 4, 5, 6, 8\}, N = 0 \rightarrow True \{\}$$

$$A = \{2, 4, 5, 6, 8\}, N = 3 \rightarrow False$$

Hint: similar to 0-1 knapsack. Think of A as set of items, N as knapsack capacity.

#### Longest common subsequence

Given two strings x and y, find the longest common subsequence and its length.

Example:

x = "ABCBDAB"

y = "BDCABA"

longest common subsequence = "BCBA"

longest common subsequence length = 4

x = "ABBACQ"

y = "XAYZMBNNALQCTRQ"

longest common subsequence = "ABACQ"

longest common subsequence length = 5

Hint: CLRS 15.4

```
LCS-LENGTH(X, Y)
    m = X.length
 1
    n = Y.length
    let b[1..m, 1..n] and c[0..m, 0..n] be new tables
    for i = 1 to m
 5
        c[i, 0] = 0
 6
    for j = 0 to n
 7
        c[0, j] = 0
 8
    for i = 1 to m
 9
        for j = 1 to n
10
             if x_i == y_i
                 c[i, j] = c[i-1, j-1] + 1
11
12
                 b[i, j] = "\\"
13
             elseif c[i - 1, j] \ge c[i, j - 1]
                 c[i,j] = c[i-1,j]
14
                 b[i,j] = "\uparrow"
15
             else c[i, j] = c[i, j - 1]
16
                b[i,j] = "\leftarrow"
17
18
    return c and b
PRINT-LCS(b, X, i, j)
     if i == 0 or j == 0
 1
2
          return
3
    if b[i, j] == "\\"
4
          PRINT-LCS(b, X, i - 1, j - 1)
5
          print x_i
     elseif b[i, j] == "\uparrow"
6
          PRINT-LCS(b, X, i - 1, j)
7
8
     else PRINT-LCS(b, X, i, j - 1)
```