Dynamic Programming

PROBLEM 01. Fibonacci number

Pseudocode (Naive Recursion):

Function Fib(n):
if n==0 **or** n==1 **then**
    return n
end if
return Fib(n-1) + Fib(n-2)
	Time complexity: O(2^n)
	Space complexity: O(1)

DP Pseudocode (Memoization/Top-down):

Create array F[0…n] = NIL
F[0]=0, F[1] = 1
 	
Function Fib(n):
if n==0 **or** n==1 **then**
    return n
end if
if F[n-1] = NIL **then**
    F[n-1] = Fib(n-1)
end if
if F[n-2] = NIL **then**
    F[n-2] = Fib(n-2)
end if
return F[n-1]+F[n-2]
	Time complexity: O(n)
	Space complexity: O(n)

DP Pseudocode (Tabulation/Bottom-up):

Function Fib(n):
Create array F[0…n]
F[0]=0, F[1] = 1
for i = 2 to n **do**
    F[i] = F[i-1]+F[i-2]
end for
return F[n]
	Time complexity: O(n)
	Space complexity: O(n)

DP Pseudocode (Problem-specific optimization):

Function Fib(n):
f0=0, f1=1
for i = 2 to n **do**
    f2 = f0+f1
    f0 = f1
    f1 = f2
end for
return f1
	Time complexity: O(n)
	Space complexity: O(1)

PROBLEM 02. 0-1 Knapsack

The weights and values of **n** items are given. *The items are not divisible, i.e., you cannot take a fraction of an item.* You have a knapsack to carry those items, whose weight capacity is **W**. Due to the capacity limit of the knapsack, it might not be possible to carry all the items at once. In that case, pick items such that the profit (total values of the taken items) is maximized.

Write a program that takes the weights and values of **n** items, and the capacity **W** of the knapsack from the user and then finds the items which would maximize the profit using a dynamic programming algorithm.

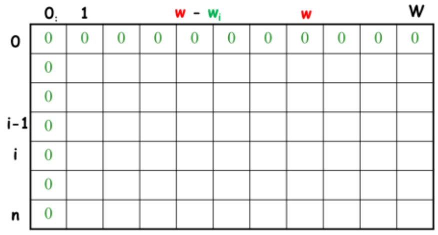
sample input	**sample output**
n weight, value … W	
4 4 20 3 9 2 12 1 7 5	item 1: 4.0 kg 20.0 taka item 4: 1.0 kg 7.0 taka profit: 27 taka

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

Pseudocode (Tabulation method):

```
Function Knapsack( v[], w[], W):
        for w = 0 to W
           P[0, w] = 0
        for i = 0 to n
           P[i, 0] = 0
           for w = 0 to W
              if w_i <= w // item i can be part of the solution
                 if (v_i + P[i-1, w-w_i] > P[i-1, w])
                       P[i, w] = v_i + P[i-1, w- w_i]
                 else
                       P[i, w] = P[i-1, w]
              else P[i, w] = P[i-1, w]  // w_i > w
```

Running time: $O(n*W)$



Time complexity: O(nW)
Space complexity: O(nW)

Food for thought:

How to print the items taken?

PROBLEM 03. Longest common subsequence

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

Example:

x = “A**BCB**D**A**B”

y = “**B**D**C**A**BA**”

longest common subsequence = “BCBA”

longest common subsequence length = 4

x = “**AB**B**ACQ**”

y = “X**A**YZM**B**NN**A**LQ**C**TR**Q**”

longest common subsequence = “ABACQ”

longest common subsequence length = 5

Hint: CLRS 15.4

```
LCS-LENGTH(X, Y)
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i − 1, j − 1] + 1
            b[i, j] = "↖"
        elseif c[i − 1, j] ≥ c[i, j − 1]
            c[i, j] = c[i − 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j − 1]
            b[i, j] = "←"
return c and b
```

```
PRINT-LCS(b, X, i, j)
if i == 0 or j == 0
    return
if b[i, j] == "↖"
    PRINT-LCS(b, X, i − 1, j − 1)
    print x_i
elseif b[i, j] == "↑"
    PRINT-LCS(b, X, i − 1, j)
else PRINT-LCS(b, X, i, j − 1)
```

PROBLEM 04. Coin change problem

Consider the problem of making change for **M** cents using the fewest number of coins. There are **d** types of coins **C = {c1, c2, …, cd}**, each coin's value is an integer and there are an infinite number of coins for each coin type. Write a greedy algorithm to make change consisting of coins in **C**.

$$\text{minNumCoins}(M) = \min \text{ of } \begin{cases} \text{minNumCoins}(M-c_1) + 1 \\ \text{minNumCoins}(M-c_2) + 1 \\ \quad \ldots \\ \text{minNumCoins}(M-c_d) + 1 \end{cases}$$

Pseudocode (Naive Recursion):

```
Function Recursive-Change (M, C, d):
    if M = 0 then
        return 0
    end if
    mnc = infinity // minimum number of coins
    for i = 1 to d do
        if C[i] <= M then
            nc = Recursive-Change(M-C[i], C, d)
            if nc+1 < mnc then
                mnc = nc+1
            end if
        end if
    end for
    return mnc
```

Pseudocode (Tabulation method):

```
Function Coin-Change (M, C, d):
    create an array mnc[0…M]
    mnc[0] = 0
    for m = 1 to M do
        mnc[m] = infinity
        for i=1 to d do
            if C[i] <= m and mnc[m-C[i]]+1 < mnc[m] then
                mnc[m] = mnc[m-C[i]]+1
            end if
        end for
    end for
    return mnc[M]
```

Food for thought:

How to print the coins taken?

Alternative Problem 2.2

Consider the problem of making a **M** meter long rope using smaller ropes. There are **d** types of ropes **C = {c1, c2, …, cd}**, each rope's value is an integer and there are an infinite number of ropes for each rope type. Joining two ropes together costs X dollar. Write an algorithm to make the **M** meter long rope with minimum costs.

PROBLEM 05. 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 → True {4, 5, 6}

A = {2, 4, 5, 6, 8}, N = 0 → True {}

A = {2, 4, 5, 6, 8}, N = 3 → False

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

Food for thought:

How to print the numbers taken?

PROBLEM 06. Rod cutting

The ***rod-cutting problem*** is the following. Given a rod of length n inches and a table of prices p_i for $i = 1, 2, \ldots, n$, 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,

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Pseudocode (Naive recursion):

```
CUT-ROD(p, n)
if n == 0
    return 0
q = −∞
for i = 1 to n
    q = max(q, p[i] + CUT-ROD(p, n − i))
return q
```

Pseudocode (Tabulation method):

```
BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] be a new array
r[0] = 0
for j = 1 to n
    q = −∞
    for i = 1 to j
        q = max(q, p[i] + r[j − i])
    r[j] = q
return r[n]
```

Pseudocode (Tabulation method):

```
Function Coin-Change (p, n):
    create an array r[0…n]
    r[0] = 0
    for rodlen = 1 to n do
        r[rodlen] = - ∞
        for cutlen = 1 to rodlen do
            if p[cutlen]+r[rodlen-cutlen] > r[rodlen] then
                r[rodlen] = p[cutlen]+r[rodlen-cutlen]
            end if
        end for
    end for
    return r[n]
```

Food for thought:

How to print the rod cuts?