

# Dynamic Programming

## PROBLEM 01. Fibonacci number

### Pseudocode (Naive Recursion):

<b>Function</b> Fib(n): 1. <b>if</b> n==0 <b>or</b> n==1 <b>then</b> 2. <b>return</b> n 3. <b>end if</b> 4. <b>return</b> Fib(n-1) + Fib(n-2)	<b>Time complexity:</b> $O(2^n)$ <b>Space complexity:</b> $O(1)$
---	---

### DP Pseudocode (Memoization/Top-down):

Create array F[0...n] = NIL F[0]=0, F[1] = 1  <b>Function</b> Fib(n): 1. <b>if</b> n==0 <b>or</b> n==1 <b>then</b> 2. <b>return</b> n 3. <b>end if</b> 4. <b>if</b> F[n-1] = NIL <b>then</b> 5.       F[n-1] = Fib(n-1) 6. <b>end if</b> 7. <b>if</b> F[n-2] = NIL <b>then</b> 8.       F[n-2] = Fib(n-2) 9. <b>end if</b> 10. <b>return</b> F[n-1]+F[n-2]	<b>Time complexity:</b> $O(n)$ <b>Space complexity:</b> $O(n)$
---	---

### DP Pseudocode (Tabulation/Bottom-up):

<b>Function</b> Fib(n): 1. Create array F[0...n] 2. F[0]=0, F[1] = 1 3. <b>for</b> i = 2 to n <b>do</b> 4.       F[i] = F[i-1]+F[i-2] 5. <b>end for</b> 6. <b>return</b> F[n]	<b>Time complexity:</b> $O(n)$ <b>Space complexity:</b> $O(n)$
---	---

### DP Pseudocode (Problem-specific optimization):

<b>Function Fib(n):</b> 1. f0=0, f1=1 2. <b>for</b> i = 2 to n <b>do</b> 3.     f2 = f0+f1 4.     f0 = f1 5.     f1 = f2 6. <b>end for</b> 7. <b>return</b> f1	<b>Time complexity: O(n)</b> <b>Space complexity: O(1)</b>
---	---

### 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  $w_i \leq w$  // 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:  $O(n*W)$**

[illegible]

**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:*

$x = \text{"ABCBDAB"}$

$y = \text{"BDCABA"}$

longest common subsequence = "BCBA"

longest common subsequence length = 4

$x = \text{"ABBACQ"}$

y = "X**A**YZ**M**BNN**A**LQ**C**TR**Q**"

longest common subsequence = "ABACQ"

longest common subsequence length = 5

Hint: CLRS 15.4

LCS-LENGTH( $X, Y$ )

```
1   $m = X.length$ 
2   $n = Y.length$ 
3  let  $b[1..m, 1..n]$  and  $c[0..m, 0..n]$  be new tables
4  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  $x_i == y_j$ 
11              $c[i, j] = c[i - 1, j - 1] + 1$ 
12              $b[i, j] = \nwarrow$ 
13         elseif  $c[i - 1, j] \geq c[i, j - 1]$ 
14              $c[i, j] = c[i - 1, j]$ 
15              $b[i, j] = \uparrow$ 
16         else  $c[i, j] = c[i, j - 1]$ 
17              $b[i, j] = \leftarrow$ 
18 return  $c$  and  $b$ 
```

PRINT-LCS( $b, X, i, j$ )

```
1  if  $i == 0$  or  $j == 0$ 
2      return
3  if  $b[i, j] == \nwarrow$ 
4      PRINT-LCS( $b, X, i - 1, j - 1$ )
5      print  $x_i$ 
6  elseif  $b[i, j] == \uparrow$ 
7      PRINT-LCS( $b, X, i - 1, j$ )
8  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**.

$$\text{minNumCoins}(M) = \min_{\text{of}} \left\{ \begin{array}{l} \text{minNumCoins}(M-c_1) + 1 \\ \text{minNumCoins}(M-c_2) + 1 \\ \dots \\ \text{minNumCoins}(M-c_d) + 1 \end{array} \right.$$

### Pseudocode (Naive Recursion):

**Function** Recursive-Change (M, C, d):

```
1. if M = 0 then
2.     return 0
3. end if
4. mnc = infinity // minimum number of coins
5. for i = 1 to d do
6.     if C[i] <= M then
7.         nc = Recursive-Change(M-C[i], C, d)
8.         if nc+1 < mnc then
9.             mnc = nc+1
10.        end if
11.    end if
12. end for
13. return mnc
```

### Pseudocode (Tabulation method):

**Function** Coin-Change (M, C, d):

```
1. create an array mnc[0...M]
2. mnc[0] = 0
3. for m = 1 to M do
4.     mnc[m] = infinity
5.     for i=1 to d do
6.         if C[i] <= m and mnc[m-C[i]]+1 < mnc[m] then
7.             mnc[m] = mnc[m-C[i]]+1
8.         end if
9.     end for
10. end for
11. 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  $C = \{c_1, c_2, \dots, c_d\}$ , 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:*

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

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

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

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  $i = 1, 2, \dots, n$ , 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( $p, n$ )
1  if  $n == 0$ 
2      return 0
3   $q = -\infty$ 
4  for  $i = 1$  to  $n$ 
5       $q = \max(q, p[i] + \text{CUT-ROD}(p, n - i))$ 
6  return  $q$ 
```

### Pseudocode (Tabulation method):

```
BOTTOM-UP-CUT-ROD( $p, n$ )
1  let  $r[0..n]$  be a new array
2   $r[0] = 0$ 
3  for  $j = 1$  to  $n$ 
4       $q = -\infty$ 
5      for  $i = 1$  to  $j$ 
6           $q = \max(q, p[i] + r[j - i])$ 
7       $r[j] = q$ 
8  return  $r[n]$ 
```

### Pseudocode (Tabulation method):

```
Function Coin-Change ( $p, n$ ):
1.  create an array  $r[0..n]$ 
2.   $r[0] = 0$ 
3.  for rodlen = 1 to  $n$  do
4.       $r[\text{rodlen}] = -\infty$ 
5.      for cutlen = 1 to rodlen do
6.          if  $p[\text{cutlen}] + r[\text{rodlen} - \text{cutlen}] > r[\text{rodlen}]$  then
7.               $r[\text{rodlen}] = p[\text{cutlen}] + r[\text{rodlen} - \text{cutlen}]$ 
8.          end if
9.      end for
10. end for
11. return  $r[n]$ 
```

### Food for thought:

How to print the rod cuts?