Fundamentals of Algorithms
CS502-Spring 2011
SOLUTION ASSIGNMENT #3

Deadline

Your assignment must be uploaded/submitted at or before **15th June 2011**

Uploading instructions

Please view the **assignment submission process** document provided to you by the Virtual University to upload the assignment.

Rules for Marking

It should be clear that your assignment will not get any credit if:

- The assignment is submitted after due date.
- The submitted assignment does not compile or run.
- **The assignment is copied.**

Objectives

This assignment will help you to understand the concepts of Knapsack Problem and Chain matrix Multiplication which results in efficient calculation time wise.

Guidelines

1. In order to attempt this assignment you should have full command on Lecture # 19 to Lecture # 26
2. In order to solve this assignment you have strong concepts about following topics
   - ✓ Chain Matrix Multiplication
   - ✓ Knapsack Problem

Recommended book for solving assignment

Cormen, Leiserson, Rivest, and Stein (CLRS) 2001, **Introduction to Algorithms**, (2nd ed.) McGraw Hill.

Estimated Time 4 hours

To understand the theme of both questions 90 minutes.Question1 solution implementation maximum time is 90 minutes and for Question2 solution implementation maximum time is one hour. It all depends upon your sheer concentration and devotion towards your lecture listening.

Question# 1 **(10)**

Consider the chain matrix multiplication for 4 matrices:

$A1$. $A2$. $A3$. $A4$
(5×6) (6×3) (3×7) (7×10)

Compute the cost table *m* in the dynamic programming algorithm for the chain matrix multiplication

Solution 1

Basic Points toward solution:

Following Recursive formulation will be used to device the solution:

$$m[i,i] = 0$$

$$m[i,j] = \min_{i \le k < j} (m[i,k] + m[k+1,j] + p_{i-1} p_k p_j)$$

Main diagonal will be filled with the base case

0			
	0		
		0	
			0

First super diagonal

m[1,2]=m[1,1] + m[2,2] + p0.p1.p2 = 0+0+5.6.3 = 90
m[2,3]=m[2,2] + m[3,3] + p1.p2.p3 = 0+0+6.3.7 = 126
m[3,4]=m[3,3] + m[4,4] + p2.p3.p4 = 0+0+3.7.10 = 210

0	**90**		
	0	**126**	
		0	**210**
			0

Second super diagonal

m[1,3]=m[1,1] + m[2,3] + p0.p1.p3 = 0+126+5*6*7 = 336
m[1,3]=m[1,2] + m[3,3] + p0.p2.p3 = 90+0+5*3*7 = 195
Minimum [1,3] = 160

for m[2,4]
m[2,4]=m[2,2] + m[3,4] + p1.p2.p4 = 0+210+6*3*10 =390
m[2,4]=m[2,3] + m[4,4] + p1.p3.p4 = 126+0+6*7.*10 = 546
Minimum for m[2,4] = 390

0	**90**	**195**	
	0	**126**	**390**
		0	**210**
			0

Third super diagonal

m[1,4]=m[1,1] + m[2,4] + p0.p1.p4 = 0+390+5*6*10 = 690
m[1,4]=m[1,2] + m[3,4] + p0.p2.p4 = 90+210+5*3*10 = 450
m[1,4]=m[1,3] + m[4,4] + p0.p3.p4 = 160+0+5*7*10 =510
Minimum for m[1,4] = 450

Resultant is,

0	**90**	**195**	**450**
	0	**126**	**390**
		0	**210**
			0

Question# 2 **(10)**

Recall that a dynamic programming solution to the 0-1 knapsack problem can be derived from the following recurrence formula for $c[i,w]$, the value of the solution for items 1, . . . , i and maximum weight w.

$$c[i,w] = \begin{cases} 0 & \text{if } i = 0 \text{ or } w = 0, \\ c[i-1,w] & \text{if } w_i > w, \\ \max(v_i + c[i-1, w-w_i], c[i-1,w]) & \text{if } i > 0 \text{ and } w \geq w_i. \end{cases}$$

In the following example the inputs are $n = 9, W = 15$, with values v_i and weights w_i:

i	1	2	3	4	5	6	7	8	9
$\mathbf{v_i}$	15	13	12	18	20	8	13	19	22
$\mathbf{w_i}$	2	3	2	3	3	1	5	2	5

Run knapsack algorithm on this table to determine the maximum value that thief may take, also mention which items the thief should take to achieve the maximum value

Solution2

Some basic thinking points to solve smoothly:

First of all here one assumption is taken for clarity purpose:

In recurrence "w_i" is current weight under consideration and second "w"in recurrence is replaced as "J" just to make the idea clear and to make difference between two. More here J/w represents the capacity of knapsack. At end J=W means maximum capacity which we have here it is "15".

First think about the recurrence which is maximizing our profit at each cell as we are making optimized decision to fill the cell**. First case** in recurrence is trivial as nothing to select for weight "0" and capacity "0". **Second case** is again not difficult; if weight under consideration "w_i" is greater than capacity "J "you have to pick the very last row value in the same column calculated already .**Last case** of recurrence is selecting the maximum value from two .In which first value is in same column very last row and second value is calculated by adding the current value to the value which is actually at very last row having column index =J-w_i **which is the core point here to understand and "maximum value of two" will be selected for the cell.**

Simple hint : You are maximizing at each cell which ever you have up to Jth Limit/capacity of your knapsack .We have maximum J=W=15 so at end we are interested in to find the maximum profit for J=W=15 capacity that is our goal .

Calculation Table :

Weight Limit (J)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$w_1 = 2$ $v_1 = 15$	0	0	15	15	15	15	15	15	15	15	15	15	15	15	15	15
$w_2 = 3$ $v_2 = 13$	0	**0**	**15**	**15**	**15**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**	**28**
$w_3 = 2$ $v_3 = 12$	0	**0**	**15**	**15**	**27**	**28**	**28**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**
$w_4 = 3$ $v_4 = 18$	0	**0**	**15**	**18**	**27**	**33**	**33**	**45**	**46**	**46**	**58**	**58**	**58**	**58**	**58**	**58**
$w_5 = 3$ $v_5 = 20$	0	**0**	**15**	**20**	**27**	**35**	**38**	**47**	**53**	**53**	**65**	**66**	**66**	**78**	**78**	**78**
$w_6 = 1$ $v_6 = 8$	0	**8**	**15**	**23**	**28**	**35**	**43**	**47**	**55**	**61**	**65**	**73**	**74**	**78**	**86**	**86**
$w_7 = 5$ $v_7 = 13$	0	**8**	**15**	**23**	**28**	**35**	**43**	**47**	**55**	**61**	**65**	**73**	**74**	**78**	**86**	**86**
$w_8 = 2$ $v_8 = 19$	0	**8**	**19**	**27**	**34**	**42**	**47**	**54**	**62**	**66**	**74**	**80**	**84**	**92**	**93**	**97**
$w_9 = 5$ $v_9 = 22$	0	**8**	**19**	**27**	**34**	**42**	**47**	**54**	**62**	**66**	**74**	**80**	**84**	**92**	**93**	**97**

Selected Items (1, 2, 3, 4, 5, 8) =>w1+w2+w3+w4+w5+w8=15=W and value 97; which is maximum.