Comp 496ALG Fall 2015

Computer Project #3 Knapsack Problem

Points: 35

Due: Dec 3

**Knapsack Problem Statement**: Given n items each with a positive integer weight and a positive integer value and given a knapsack that holds at most weight W, fill the knapsack with a subset of the n items so that (1) the total weight of the items <=W and (2) the total value of the items is maximized. You cannot fill the knapsack with fractional items.

In this project you will implement in Java or C++ two different methods that solve Knapsack Problem(KNP) and then you will do a comparison of the runtimes for various values of n, weights and values. All work must be your own. Do not download solutions from the internet or from other outside sources except where specifically allowed in Part 1.

Input: int n //number of items //index items from 1 to n

int[] weight // weight[i] = weight of item i+1

int[] value // value[i] = value of item i +1

int W //weight capacity of the knapsack

Output:

1. Optimal set of items to put into the knapsack or an approximation
2. Total weight of the items, Total value of the items
3. Runtime of algorithm
4. **Subset Algorithm.** Create or find an algorithm that generates all possible subsets of {0, .., n-1} or { 1, ..,n} , for any positive integer n. This should be stand-alone from the rest of your code. You can look this up in a textbook or on the internet. If you do so, give a reference to where you got the code or algorithm.
5. **Brute Force Method to Solve KNP.** (Generates optimal solution) Use your algorithm in Part 1 to generate all subsets of {0,..,n-1} or { 1, .., n} . Use it to find all subsets of items with the maximum total value that satisfy the constraint that the total weight must be <=W.
6. **Dynamic Programming Method to solve KNP. (**Generates an optimal solution**).** See class discussion and text book. You must use the dynamic programming recurrence equations from class or text book ( Equation 6.11). You must implement the dynamic programming algorithm.

OPT(0,w) = 0 for 0 <= w <= W

For i > 0,

if w < wi then OPT(i,w) = OPT(i-1,w). Otherwise

OPT(i,w) = max { OPT(i-1,w) , vi + OPT(i-1, w – wi) }

1. **Test Cases**. Test your methods using the following arrays of weights wi , values vi and knapsack capacities.

Test Case #1

int n = 6;

int[] weights = { 2,4,3, 4,4, 1};

int W = 10;

int[] values = { 1,2,3, 3,3, 6}

Test Case #2

int n = 10;

int[] weights = {3,4,2,5,6,1,2,7,8,2};

int W = 12;

int[] values = {6,7,4,3,2,6,8,7,9,6};

Test Case #3

int n = 20;

int[] weights = { 2,3,4,2,6,5,3,7,2,4,3,1,5,6,2,1,1,3,4,3};

int W = 18;

int[] values = { 2,3,4,1,2,5,3,2,4,6,2,2,1,3,4,5,6,2,1,9};

Test Case #4

int n = 25;

int[] weights = {9,16,12,8,7, 14,7,8,9,14,15,18,20,2,4,5,10,11,3,17 ,15,18,15,9,7 };

int W = 100;

int[] values = {1,7,3,4,5, 5,3,4,6,2, 6,6,4,2,1, 1,2,2,4,5,4,3,2,1,3};

OUTPUT: The computer program output format of your test cases should look like this:

Test Case#1

Number of Items = 6

Weight Capacity of Knapsack: 10

Brute Force Optimal Solutions (List all solutions):

Subset of Items = { }; Subset Weight = ; Subset Value = ;

Subset of Items = { }; Subset Weight = ; Subset Value = ;

…

Runtime ( ms) =

Dynamic Programming Optimal Solution (List one solution):

Subset of Items = { }; Subset Weight = ; Subset Value = ;

Runtime ( ms) =

Your Subsets of Items should be formatted as { 2,4,6,7} to indicate items 2,4,6 and 7 have been selected.

1. The Brute Force Method should run in O(n\* 2n) and the Dynamic Programming Method should run in O( nW) time. If W is large enough the Brute Force method may be faster than the dynamic programming method. Find an n and W where the runtime of your code take longer for the Dynamic Programming Method than the Brute Force Method. [Hint: make W dependent on n]
2. **HAND IN** (Stapled in this order)**:**
   1. Cover Page
   2. Output from Test Case #1. For this test case only, print the OPT matrix.
   3. Output from Test Case #2
   4. Output from Test Case #3
   5. Output from Test Case #4
   6. One page report (typed, separate page) on results of experiments in Item 5.
   7. Source Code with main program that produced the output from the test cases.