Homework: Greedy Algorithms

This document defines the **homework assignments** for the "Algortihms" course @ Software University. Please submit a single **`zip`** / **`rar`** / **`7z`** archive holding the solutions (source code) of all below described problems.

Problem 1. Fractional Knapsack Problem

A modification of the famous knapsack problem is the so-called continuous (or fractional) knapsack problem.

We have **N** items, each with a certain **weight** and **price**. The knapsack has a **maximum capacity**, so we need to choose what to take in order to **maximize the value (price)** of the items in it.

Unlike the classical version of the problem where an object should either be taken in its entirety or not at all, in this version **we can take a fraction of each item**. For example, such items may be liquids or powders; unlike solid objects which (presumably) we cannot split, we'll assume that all items under consideration can be divided in any proportion. Therefore, the weight can be thought as the maximum quantity Q of an item we are allowed to take – we can take any amount in the range [0 … Q]. Note that in this version of the problem the knapsack will always be filled completely (if the total quantity of items is greater than its capacity).

Items will be given on separate lines in format **price -> weight**. Round all numbers to two digits after the decimal separator. Examples:

Input	Output

`Capacity: 16`
`Items: 3`
`25 -> 10`
`12 -> 8`
`16 -> 8`
	`Take 100% of item with price 25.00 and weight 10.00`
	`Take 75.00% of item with price 16.00 and weight 8.00`
	`Total price: 37.00`

`Capacity: 13`
`Items: 2`
`13 -> 7`
`15 -> 7`
	`Take 100% of item with price 15.00 and weight 7.00`
	`Take 85.71% of item with price 13.00 and weight 7.00`
	`Total price: 26.14`

`Capacity: 22`
`Items: 4`
`50 -> 25`
`34 -> 25`
`41 -> 25`
`3 -> 25`
	`Take 88.00% of item with price 50.00 and weight 25.00`
	`Total price: 44.00`

`Capacity: 134`
`Items: 9`
`12 -> 14`
`45 -> 54`
`98 -> 78`
`21 -> 51`
`64 -> 11`
`90 -> 117`
`33 -> 17`
`64 -> 23`
`7 -> 3`
	`Take 100% of item with price 64.00 and weight 11.00`
	`Take 100% of item with price 64.00 and weight 23.00`
	`Take 100% of item with price 7.00 and weight 3.00`
	`Take 100% of item with price 33.00 and weight 17.00`
	`Take 100% of item with price 98.00 and weight 78.00`
	`Take 14.29% of item with price 12.00 and weight 14.00`
	`Total price: 267.71`

Hint: use a straightforward greedy algorithm: take the best item (max price / weight) as much as possible, then the next best item, etc.

Problem 2. Processor Scheduling

We are given a processor capable of performing only **one task at a time** and **N** tasks. Each task has two properties associated with it – **value v** of completing the task and a **deadline d**. Each task takes exactly one unit of time to complete and should be completed at or before its deadline. E.g., if we have a task with deadline 2, we can complete it either at step 1 or step 2, but not afterwards. Given a set of tasks numbered from 1 to N, find a schedule which will **maximize the total value** of the tasks performed.

Consider the following scenario:

Task number	1	2	3	4	5
Value	40	30	15	20	50
Deadline	1	2	1	1	2

With a maximum of 2 steps (largest deadline is 2) we can complete at most two tasks. Performing one of the tasks that has a deadline 2 on the first step leaves us with no choice for the second step, but to take the other task with deadline 2 (tasks with deadline 1 can no longer be performed). Therefore, 2 -> 5, or 5 -> 2 will produce a total value of 30 + 50 = 80.

It is obvious from the table above that the optimal solution is 1 -> 5 which produces total value of 40 + 50 = 90.

Since tasks can be performed in different order, when printing the output, order the tasks based on their deadline in increasing order and then by value in decreasing order.

Examples:

Input	Output
Tasks: 5 40 – 1 30 – 2 15 – 1 20 – 1 50 – 2	Optimal schedule: 1 -> 5 Total value: 90
Tasks: 3 25 – 1 14 – 1 43 - 1	Optimal schedule: 3 Total value: 43
Tasks: 6 5 – 3 6 – 4 2 – 1 3 – 4 8 – 2 4 – 3	Optimal schedule: 5 -> 1 -> 6 -> 2 Total value: 23 2 -> 1 -> 5 -> 3

Hint: At each step, take the task with highest value. Make sure that when adding the task to the result, all tasks that were selected can be completed.

Problem 3. Knight's Tour

Given a board of size **NxN** (a standard square matrix), a chess knight can perform a tour of the board, **visiting each cell only once**. The knight moves according to the rules of chess (in an L-shaped pattern) and starts from the upper-

left corner. Write a program which finds and prints the path the knight needs to take in order to visit all cells. From the input you receive just the board size **N** (N > 4).

On the output, print the board where each cell's value is the number of the step which led to it, e.g. cell (0, 0) will be 1, the first step, the next visited cell (0, 2) will have value 2, etc. In order to format the output, assume NxN < 1000 – no cell's value will be more than 3 digits (you can pad the values with spaces with the string.PadLeft() method).

Examples:

Input	Output
5	1 12 25 18 3 22 17 2 13 24 11 8 23 4 19 16 21 6 9 14 7 10 15 20 5
6	1 32 9 18 3 34 10 19 2 33 26 17 31 8 25 16 35 4 20 11 36 27 24 15 7 30 13 22 5 28 12 21 6 29 14 23
12	1 24 61 96 3 26 91 110 5 28 31 112 62 95 2 25 92 141 4 27 114 111 6 29 23 60 93 144 97 90 135 142 109 30 113 32 94 63 98 89 140 143 130 115 136 119 108 7 59 22 139 100 129 134 137 126 107 116 33 118 64 99 88 133 138 127 106 131 120 125 8 81 21 58 65 128 101 132 121 124 105 82 117 34 46 55 102 87 66 77 104 83 122 75 80 9 57 20 47 54 103 86 123 76 79 84 35 74 42 45 56 67 50 53 78 85 70 73 10 13 19 48 43 40 17 68 51 38 15 12 71 36 44 41 18 49 52 39 16 69 72 37 14 11

Hint: Use Warnsdorff's rule to decide which cell the knight should visit next.

Problem 4. Best Lectures Schedule

A long time ago in a neighborhood far, far away SoftUni used to have just one lecture hall. Let's have a list of lectures, each having a **start time s** and a **finish time f** (**s** and **f** will be positive integers). Obviously, only one lecture can be presented at a time, they cannot overlap. Write a program that maximizes the number of lectures presented.

Lectures will be given in format **name: start – finish**. Assume the name will contain only word characters (letters, digits and '_'). Examples:

Input	Output
Lectures: 4 Java: 1 - 7 OOP: 3 - 13 C_Programming: 5 - 9 Advanced_JavaScript: 10 - 14	Lectures (2): 1-7 -> Java 10-14 -> Advanced_JavaScript
Lectures: 3 Programming_Basics: 3 - 5	Lectures (1): 2-4 -> PHP

PHP: 2 – 4 Photoshop: 1 – 6	
Lectures: 7 Advanced_CSharp: 3 – 8 High_Quality_Code: 7 – 10 Databases: 5 – 12 ASP_NET: 9 – 14 Angular_JS: 13 – 15 Algorithms: 15 – 19 Programming_Basics: 17 – 20	Lectures (3): 3-8 -> Advanced_CSharp 9-14 -> ASP_NET 15-19 -> Algorithms

Hint: Use the finish time to decide which lecture should be chosen at each step. After a lecture is chosen, remove all others that overlap with it.

Problem 5. Egyptian Fractions

In mathematics, a fraction is the rational number **p/q** where **p** and **q** are integers. An Egyptian fraction is a sum of fractions, each with **numerator 1** where **all denominators are different**, e.g. 1/2 + 1/3 + 1/16 is an Egyptian fraction, but 1/3 + 1/3 + 1/5 is not (repeated denominator 3).

Every positive fraction (q != 0, p < q) can be represented by an Egyptian fraction, for instance, 43/48 = 1/2 + 1/3 + 1/16. Given **p** and **q**, write a program to represent the fraction **p/q** as an Egyptian fraction.

Examples:

Input	Output
43/48	43/48 = 1/2 + 1/3 + 1/16
3/7	3/7 = 1/3 + 1/11 + 1/231
23/46	23/46 = 1/2
22/7	Error (fraction is equal to or greater than 1)
134/3151	134/3151 = 1/24 + 1/1164 + 1/2445176

Note: There may be more than one correct solution, e.g. 3/7 = 1/4 + 1/8 + 1/19 + 1/1064.

Hint: You can complete the expression by starting with the biggest fraction with numerator 1 which added to the expression keeps it smaller than or equal to the target fraction. The biggest fraction is the one with smallest denominator – 1/2. Increase the denominator until you've found a solution.