COSC 4P03 – Advanced Algorithms
Winter 2019
Assignment #1

Due Date: 11th February, Noon Late Date: 14th February, noon
This assignment accounts for 15% of your final grade and is worth a total of 150 marks.

Question 1 – "Carnival Drop" (50 marks)

You are at a carnival, faced with the problem of trying to win enough tokens to pay for the outrageously expensive food. You have come across a game that you think might be winnable for a smart Computer Science student such as yourself. The game consists of dropping a marble from the top of a *n*x*n* board towards the bottom. At each step, you can shift the board slightly so that when the marble travels to the level below it can either drop directly down (i.e. in the same column), or it can move one column to the left or to the right. Each of these possible moves has an associated integer value, and some of these values may be negative. The goal is to find a path that gives the maximum possible total value – if you are able to follow such a path then you win the game and are given enough tokens to pay for your dinner.

a) [8] Show that this problem exhibits optimal substructure.
b) [8] Give a recursive definition of the problem.
c) [10] Write a dynamic programming algorithm to solve the above problem. The algorithm should be written at approximately the same level of detail as those in the main reference book, CLRS ("Introduction to Algorithms", by T.Cormen, C.Leiserson, R.Rivest & C.Stein).
d) [24] Write a program incorporating the above algorithm.

Input:
Input is to be from a text file consisting of several scenarios.
The first line of the text file contains a single integer *m*, specifying the number of scenarios.
Each scenario will be described as follows:

- One line containing a single integer *n*, specifying an *n x n* board.
- A sequence of *(n-1)*n* lines each containing up to 3 values, specifying for each location *x* in the first *n*-1 rows the value *p(x,y)* of moving from *x* to each valid choice *y* in the level below. These are given in row-major order. Therefore:
  - The first line specifies *p((1,1), (2,t))* for each valid column *t* from left to right.
  - …
  - The *n*'th line specifies *p((1,n), (2,t))* for each valid column *t* from left to right.
  - The *(n+1)*'th line specifies *p((2,1), (3,t))* for each valid column *t* from left to right.
  - …
  - The *((n-1)*n)*'th line specifies *p((n-1,n), (n,t))* for each valid column *t* from left to right.
  - For all of the above, if it is not legal to move from location *x=(i,s)* to location *y=(i+1,t)* then *p((i,s), (i+1,t))* should be set to -1000, although this is not included in the input file.

Output:
Output is to be to a text file.

For each scenario, print a sequence of locations to visit that will give you as many dollars as possible, and the total value you earned using that sequence. There may be more than one optimal choice; you are only required to print *one* such choice.

You must submit the results of running your program on the input file available from the course web page.

Example:

The following scenario:

```
3
1 -1
2 3 4
1 3
2 2
5 3 1
2 3
```

Should generate the following output:

```
Locations visited: (1,2) (2,2) (3,1)
Total earned: 8
```

Question 2 – "Pretty Printing" (50 marks)

This question is adapted from question 15-4 on page 405 of the main reference book, CLRS ("Introduction to Algorithms", by T.Cormen, C.Leiserson, R.Rivest & C.Stein) together with the problem "Formatting Text" from the programming contest problem archive at the UVa Online Judge – see: `https://uva.onlinejudge.org/external/7/709.pdf`.

In this question you are to write a program to format text "nicely". The formatting consists of inserting blanks into text, such that the result satisfies the following definition of "nice":

- All lines (including the last) are to have the same length, measured in number of characters.
- There must be at least one blank character between adjacent words on a line.
- The cost of a gap of k blank characters is $(k\text{-}1)^2$, and the total cost of all gaps in the text must be minimized.
- All lines must both start and end with a non-blank character, with one exception: if there is only 1 word on a line, then it should be placed at the start of the line, and the total badness of the line is given as 500.

a) [8] Show that this problem exhibits optimal substructure.
b) [8] Give a recursive definition of the problem.
c) [10] Write a dynamic programming algorithm to solve the above problem. Follow the same guidelines as for question 1, part c.
d) [24] Write a program incorporating the above algorithm.

Input:

Input is to be from a text file consisting of several paragraphs.

On the line immediately prior to each paragraph will be an integer M, $1 \le M \le 80$, which is the desired width of the paragraph. A value of M=0 indicates the end of the input.

Each word consists of characters with ASCII codes 33 to 126 (Unicode U+0021 to U+007E) inclusive. Words are separated by 1 or more spaces, or by a newline character, and paragraphs are separated by 1 or more empty lines. There are at most 500 words per paragraph.

Output:

Output is to be to a text file.

For each paragraph, write the total cost of formatting that paragraph on a line by itself. This should be immediately followed by the formatted paragraph. Paragraphs should be separated by a blank line.

You must submit the results of running your program on the input file available from the course web page.

Question 3 – "Super Greedy Knapsack" (50 marks)

In the 0-1 knapsack problem, a thief robbing a store has a knapsack that can carry at most W pounds. There are n items in the store: item i is worth v_i dollars and weighs w_i pounds. The thief must decide which items to take in order to have as valuable a load as possible. This problem is NP-complete.

Suppose that we modify the above problem so that each item has exactly the same value per pound (i.e. for every item i, $v_i = k*w_i$ for some constant k). In addition, suppose that each weight w_i is larger than the sum of all smaller weights: for example, {2, 16, 1, 4, 8} satisfies this property.

a) [10] Show that this problem has the greedy-choice property.
b) [5] Show that this problem exhibits optimal substructure.
c) [10] Consider the representation of this problem as a weighted matroid M = (S,I), where:
   - The set S is the set of items,
   - A is an independent subset of I iff its weight does not exceed W, and
   - The weight of A is defined as the sum of the weights of the items in A.

   Explain why this representation does not satisfy all of the properties of a matroid.
d) [5] Write a greedy algorithm to solve this problem.
e) [20] Write a program incorporating the above algorithm.

Input:

Input is to be from a text file consisting of several scenarios.
The first line of the text file will contain a single integer m, specifying the number of scenarios in the file.
Each scenario will be described in 2 lines as follows:

- The first line contains three integers: n, the number of items in the store, k, the value per pound of each item, and W, the maximum weight that the knapsack can hold.
- On the next line, there is a sequence of n integer values; the i'th value in the sequence specifies the weight of item i in the store.

Output:

Output is to be to a text file.

For each scenario, there should be 2 lines of output: on the first line, give the list of items chosen during an optimal theft. On the second line, give the total value of the items stolen.

You must submit the results of running your program on the input file available from the course web page.

Example:

The following scenario:

```
4 10 80
10 5 50 20
```

Should generate the following output:

```
Items stolen: 1, 3, 4
Total value: 800
```

Additional Notes:

1. Sample data will be provided on the course website, but you should expect your TA to run your programs with other data, including "tricky" input.
2. In order to simplify marking, try to adhere to the requested format as much as possible.

Submission Requirements:

All of the following must be placed in a sealed envelope in the 4P03 assignment box:

1. A cover sheet, available from `http://www.cosc.brocku.ca/forms/cover`, completely filled out. Your assignment will not be marked unless one is submitted with the assignment.
2. Commented and properly documented printouts of all source code for your programs.
3. Printouts of all output as specified above.
4. All answers to the "written" parts of this assignment.
5. Any information required to run your programs. Programs may be written in Java or C++.

You must also submit your assignment electronically for the purpose of plagiarism detection. To do this, create a directory on Sandcastle containing all files for this assignment, and run the script `submit4p03` from this directory.