CO 353 Winter 2018: Project 2

Due: February 26 at 8pm

1 Objective

The project consists in implementing the dynamic programming algorithm for the integer knapsack problem. The implementation must be *correct* and *efficient*. In particular, given a knapsack instance

$$\begin{aligned} \max \quad & \sum_{j=0}^{n-1} c_j x_j \\ \text{s.t.} \quad & \sum_{j=0}^{n-1} a_j x_j \leq b \\ & x \in \mathbb{Z}^n_+, \end{aligned}$$

the implementation must have $O(n \cdot b)$ complexity (in the arithmetic model, i.e., addition, subtraction, multiplication, division, comparison and assignment are $O(1)$).

This project can be completed in groups of 1 or 2. You can also form groups of 3, but *only* if you first get my approval by email (lpoirrier (at uwaterloo)) by February 11th. If you already had approval for a group of 3 for the previous assignment, then you are automatically allowed (but not required) to form *the same* group of 3 for this assignment (no need to contact me in this case).

2 Input and output

The input is n and the positive integer coefficients $c \in \mathbb{Z}^n$, $a \in \mathbb{Z}^n$, $b \in \mathbb{Z}$. They are given in a file of the form:

n					
c_0	c_1	c_2	$\dots$	c_{n-1}	
a_0	a_1	a_2	$\dots$	a_{n-1}	b

A complete example is given by:

4				
7	9	2	15	
3	4	1	7	10

... and corresponds to the knapsack instance:

$$\begin{aligned} \max \quad & 7x_0 + 9x_1 + 2x_2 + 15x_3 \\ \text{s.t.} \quad & 3x_0 + 4x_1 + 1x_2 + 7x_3 \leq 10 \\ & x \in \mathbb{Z}^4_+. \end{aligned}$$

The output file must describe an optimal solution to the given knapsack problem instance. It consists of a single line of the form:

x^*_0	x^*_1	$\dots$	x^*_{n-1}

In our example, a correct solution file could be:

2	1	0	0

Note that, in order to (optionally) let you include more information in the output file, everything between a # sign and the end of a line will be ignored. So, for example, the following is also correct:

```
# n = 4
2    1    0    0      # objective:  23
# some other comment
```

More example input and output files are given at `https://www.math.uwaterloo.ca/~lpoirrie/co353.html`.

3 Running the code

Your code must take exactly two arguments. The first argument is the name of a file containing the input knapsack instance. The second argument is the name of a file where the output solution is to be written.

You have a choice between the following languages for the implementation: C, C++ or Python.

3.1 In C or C++

Your code must be portable to any environment with a standards-conforming C or C++ compiler. It must be possible to compile and run it by using the following commands:

in C:

```
gcc -O3 -o knapsack knapsack.c
./knapsack input.txt output.txt
```

in C++:

```
g++ -O3 -o knapsack knapsack.cpp
./knapsack input.txt output.txt
```

In other words, you must implement your code in a file called `knapsack.c` (in C) or `knapsack.cpp` (in C++).

Note: Optionally, you may provide a `Makefile`, in which case your code will be compiled by running `make`. The resulting executable must be named `knapsack`. If you choose to do this, it is your responsibility to ensure that the `Makefile` is correct, so do it only if you are familiar with `Makefile`s already.

3.2 In Python

It must be possible to run your code by executing the following command:

```
python knapsack.py input.txt output.txt
```

In other words, you must implement your code in a file called `knapsack.py`. If you need a specific version of the Python language, please mention it in your email.

4 Submitting the project

You submit your implementation by sending a single email to both `lpoirrier (at uwaterloo)` and `wjtoth (at uwaterloo)`, by 8pm on Monday, February 26th, 2018. Late submissions will not be accepted. The subject line of your email must contain the string `CO353`. Your email must contain **a single attachment file**: an archive in

the format `.zip` or `.tgz`. The name of the archive is formed by the UWaterloo IDs of your group members, in any order, separated by underscores. For example: `jwtoth_lpoirrie.tgz`. The archive contains (at least) two files:

- a source file for your implementation, and
- a file called `notes.txt` (or `notes.md` if you prefer), see below.

The file `notes.txt` only specifies the names of the (1, 2 or 3) members of your group. No further explanations are necessary for this assignment.

5 Contribution

You can use the standard library in your language of choice. No other library is allowed. In case of doubt ask me. You can consult any source of information you want (books, internet, scientific papers, etc.), but copying code is not allowed, regardless of the source. Every single line of your code must be written by a member of the group. Each member of the group must understand (and must be able to justify) every line of the code, including the ones they did not write. After the due date, I may call an individual group member to come to my office hours; if that member does not show sufficient understanding of the code, *all* group members will get zero.

6 Grading criteria

The project will be graded on a total of 10 marks, divided in 4 categories. Pay attention to the first two: if your code does not follow the specification or is incorrect, it will not be tested for speed, and you will get zero for the fourth category (efficiency).

- Specification (1 mark). Your submission must follow the specifications given in this document (your email must have the correct recipients/subject/attachment, your source files must follow the template given above, your code must compile and take the right arguments). The rules are rigid because testing will be automated.
- Correctness (2 marks). The output must describe an optimal solution to the knapsack problem instance given in input.
- Code quality (2 marks). It should be reasonably easy to understand your code. Comments can be used to help in that regard, but with moderation. The code itself must be clear and readable.
- Efficiency (5 marks). Your code must have the prescribed complexity ($O(n \cdot b)$). It is your responsibility to understand the computational complexity of the functions and data structures that you use, even if they are part of the core language. For example, if you use an array of elements, removing an element in the middle is $O(n)$, even if it appears to be a single operation. This can be done in $O(1)$ with a linked list.