COMP319 Algorithms, Spring 2019

Programming Assignment 4

Instructor: Gil-Jin Jang (gjang@knu.ac.kr), School of Electronics Engineering

1. 0-1 knapsack problem

Knapsack problem is finding a best subset of given items that satisfies:

- the sum of the item weights is less than the capacity of the knapsack (*W*).
- the sum of the item values is maximized. If we allow 0-1 choices only, the items are indivisible, and the relaxed prob can be solved with dynamic programming.
- **To do:** Design an algorithm that finds:
  - the subset of the items by their numbers (chosen from 1 to *N*).
  - the maximum value.
- **INPUT:** The input is given by a text file, with each row represents each item's weight and value. The last line is the maximum weight followed by -1, so you can finish reading the file if the input number is -1.

  ```
  2 3
  3 4
  4 6
  5 7
  8 -1
  ```

  All the values are POSITIVE integers. You may use file I/O, or standard input. From the input file, number of items *n* = (weight, benefit) of the items are: `(2,3), (3,4), (4,6), (5,7)`, and *W* = 8 (max weight).
- **OUTPUT:**

  ```
  2 4 11
  ```

2. 0-1 knapsack with one item split

In this problem, it is assumed that ONLY ONE item can be split by HALVES --- one item is split into two items with the sa weight and the same value, half the weight and half the value, respectively. We may choose one half-split item, or whole item, whichever maximizes the value.

- **To do:** Design an algorithm that finds:
  - the subset of the items by their numbers (chosen from 1 to *N*).
  - the maximum value.
  - if necessary, specify one item to be split by halves by appending **'x0.5'** to the item number.
- **Note:** when the weights and values are odd numbers, their split weights and values may have fractional parts (0.5). I use a dynamic programming algorithm (probably), those odd number cases should be considered.
- **INPUT:** The input is given by a text file, with each row represents each item's weight and value. The last line is the maximum weight followed by -1, so you can finish reading the file if the input number is -1.

  ```
  2 3
  3 4
  4 6
  5 7
  8 -1
  ```

All the values are POSITIVE integers. You may use file I/O, or standard input. From the input file, number of items n =
(weight, benefit) of the items are: `(2,3), (3,4), (4,6), (5,7)` , and W = 8 (max weight).

- **OUTPUT:**

```
1x0.5 2 3 11.5
```

3. 0-1 knapsack with one duplicate item

Same as problem 2, except that **EXACTLY ONE ITEM** and be added **TWICE**.

- **To do:** Design an algorithm that finds:
  - the subset of the items by their numbers (chosen from 1 to N).
  - the maximum value.
  - if necessary, specify one item to be split by halves by appending **'x2'** to the item number.
- **INPUT:** The input is given by a text file, with each row represents each item's weight and value. The last line is the maximum weight followed by -1, so you can finish reading the file if the input number is -1.

```
2 3
3 4
4 6
5 7
8 -1
```

All the values are POSITIVE integers. You may use file I/O, or standard input. From the input file, number of items n =
(weight, benefit) of the items are: `(2,3), (3,4), (4,6), (5,7)` , and W = 8 (max weight).

- **OUTPUT:**

```
3x2 12
```

4. 0-1 knapsack with two identical knapsacks

Same as problem 1, except there are **TWO KNAPSACKS** to be filled.

- **To do:** Design an algorithm that finds:
  - the subset of the items by their numbers (chosen from 1 to N) and **knapsack numbers** (1 or 2).
  - the maximum value.
- **INPUT:** The input is given by a text file, with each row represents each item's weight and value. The last line is the maximum weights of the **TWO KNAPSACKS** followed by -1, so you can finish reading the file if the input number is -

```
2 3
3 4
4 6
5 7
8 7 -1
```

All the values are POSITIVE integers. You may use file I/O, or standard input. From the input file, number of items n =
(weight, benefit) of the items are: `(2,3), (3,4), (4,6), (5,7)` , and W = `(8, 7)` (max weights).

- **OUTPUT:**

```
2 1 4 1 1 2 3 2 20
```

- **Note:** may not be solvable by dynamic programming (even the instructor does not know). Consider greedy algorithr