Middle East Technical University

Department of Computer Engineering

# CENG 111

Fall 2017

Take Home Exam

## REGULATIONS

**Due date:** 23:59, 21 January 2017, Sunday *(Not subject to postpone)*

**Submission:** Electronically. You will be submitting your program source code through a text file which you will name as `the4.py` by means of the COW system.

**Team:** There is **no** teaming up. This is an EXAM.

**Cheating:** Source(s) and Receiver(s) will receive zero and be subject to disciplinary action.

## PROLOGUE: THE NEW ORDER

Please watch the prologue at the following link with speakers on:

`http://www.ceng.metu.edu.tr/~ceng111/the4-prologue.mp4`

If you are unable to watch the video, here is the text:

> "A long time ago, in a galaxy far, far away...
>
> With the death of the Supreme Leader by Ben Solo, the First Order is left in turmoil. As the new Supreme Leader, Ben Solo cornered the last remnants of the resistance and failed again because of the last sparkle from Luke Skywalker.
>
> Feeling frustrated, and not to repeat failures of the previous Supreme Leaders, Ben Solo wants to reorder and restructure the command-chain of the First Order to make a New Order that is more powerful and effective. To finish the resistance once and for all, the Supreme Leader asks your help."

## PROBLEM

*"The greatest teacher, failure is."* – Master Yoda

Ben Solo will give you the current command-chain as a *tree* (e.g., as shown in Figure 1), and you will restructure the tree so that the following constraints are satisfied:

1. **Constraint 1:** Commanders in the new order are given responsibilities based on their force strengths.

   Explanation: Ben Solo wishes this new order to be such that the force strength $f_i$ of a commander $i$ on height $h_i$ of the tree is bigger than $f_j$ for any commander $j$ on lower heights. If there are two commanders $i$ and $j$ that violate this constraint, they should be swapped.

2. **Constraint 2:** The height of the tree should be lowered.

   Explanation: After arranging the new order according to Constraint 1, Ben Solo wants the tree to be re-structured according to the height. A commander $i$ on a height below than a given threshold $T$ are placed under a commander $j$ on a higher height that is above the height threshold $T$ if (i) $j$ is on the path from the Supreme Leader to commander $i$, and (ii) $f_j \geq \sum_{k \in C_j} f_k$, where $C_j$ is the set of children of commander $j$. Note that the constraint $f_j \geq \sum_{k \in C_j} f_k$ should be satisfied after addition as well; moreover, this constraint is relevant only for the commanders under which new children are to be added. Commanders that may not be placed on a higher height are thankfully executed.

An example snapshot of the command hierarchy of the First Order is shown in Figure 1(a), where we see each commander being represented with two fields: (i) $n$: the name of the commander (a string), (ii) $f$: the strength of the force within the commander (an integer).

In this example, checking Constraint 1 reveals that commanders "Jack" and "Nico", "Fredo" and "Ela", "Vincenzo" and "Han" violate the constraint. Applying the solutions suggested by the Supreme Leader, we swap these commanders and the hierarchy is updated as shown in Figure (b).

Checking Constraint 2 with height threshold of 2 for the example hierarchy in Figure (b), we see that commanders on height 1 should be re-placed on higher heights. However, looking at the definition of the constraint, we see that only commander "Nico" can accept new children with a force strength less than or equal to 2. Here we could place either (i) two commanders with strength 1 each or (ii) a commander with strength 2. Both options are accepted by the Supreme Leader as long as no commander is retired while there was an option to place him/her under a commander as suggested by Constraint 2.

# SPECIFICATIONS

- Write a Python function named `help_ben_solo()`, which will take (i) the current command hierarchy as a tree, and (ii) the height threshold. The tree is a nested list having the following structure for a node (commander):

  `[name, f, child_1, ..., child_i]`

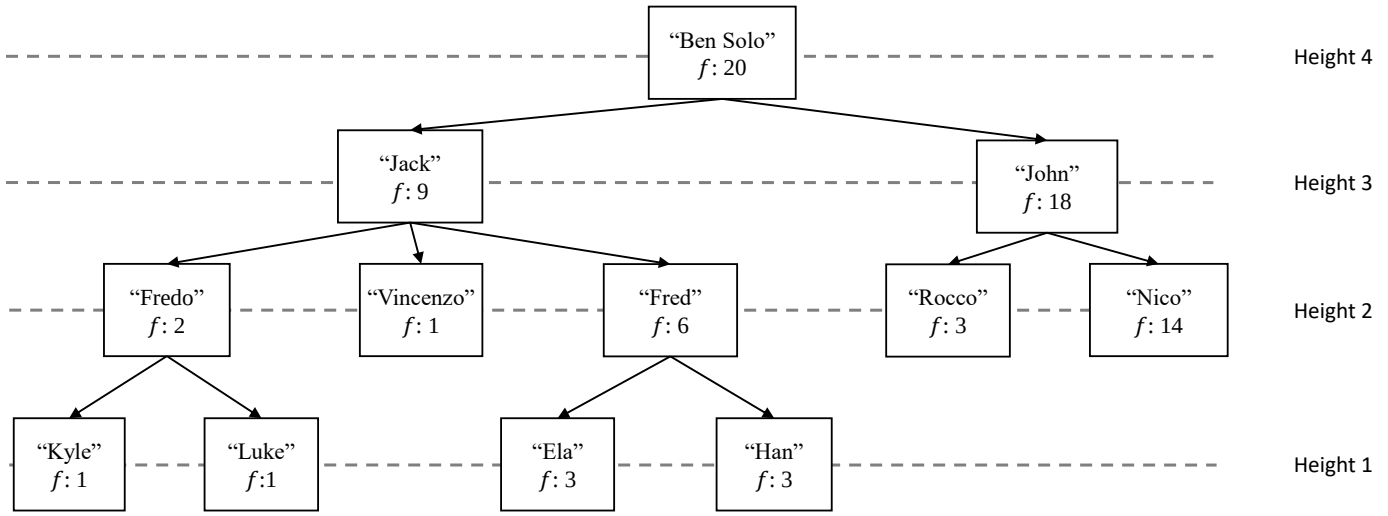
  where `name` is a string; `f` is the force strength of the commander; and `child_1, ..., child_i` are the children (sub-commanders) of the current node. If the current node is a leaf node, then the node is only represented as:

  `[name, f]`

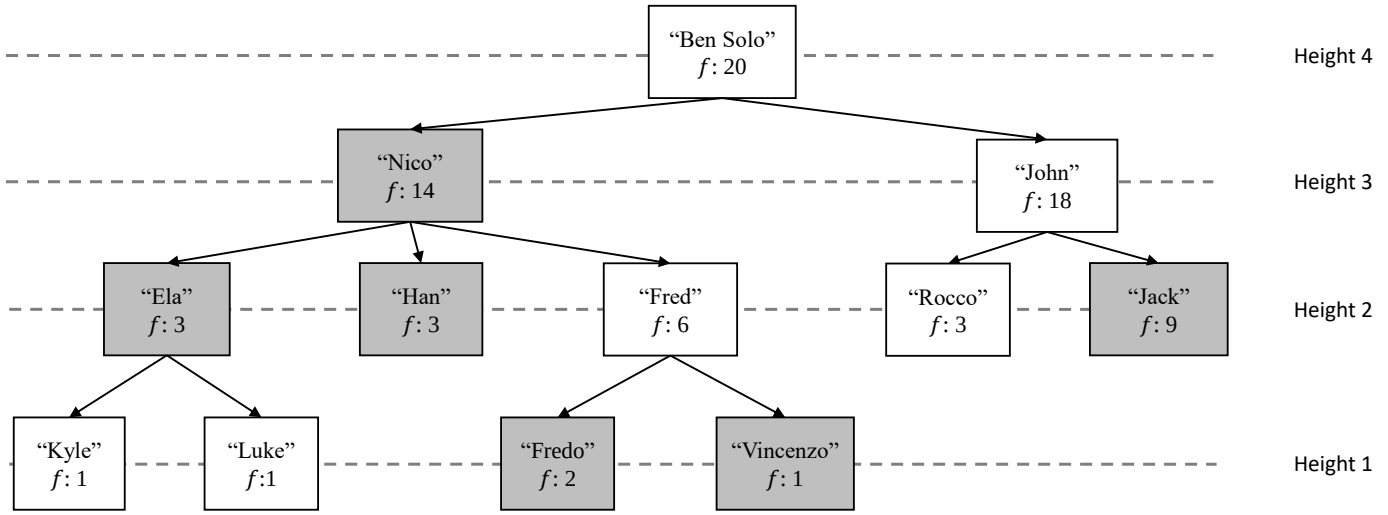
- The function should return a tuple of two lists: (i) the New Order as a tree in the format as that of the input, and (ii) the list of commanders that were retired.
- You can assume that names are case sensitive, and that each name in the command hierarchy is unique.
- We will not test your solution with erroneous inputs.
- The Supreme Leader wishes the commanders to be left as they are if they do not violate any of the constraints.
- The order within the children of a node is not important for the Supreme Leader.
- There may be more than one solution. Any solution is accepted as long as it satisfies the Supreme Leader's constraints and guidelines.

## Example Run

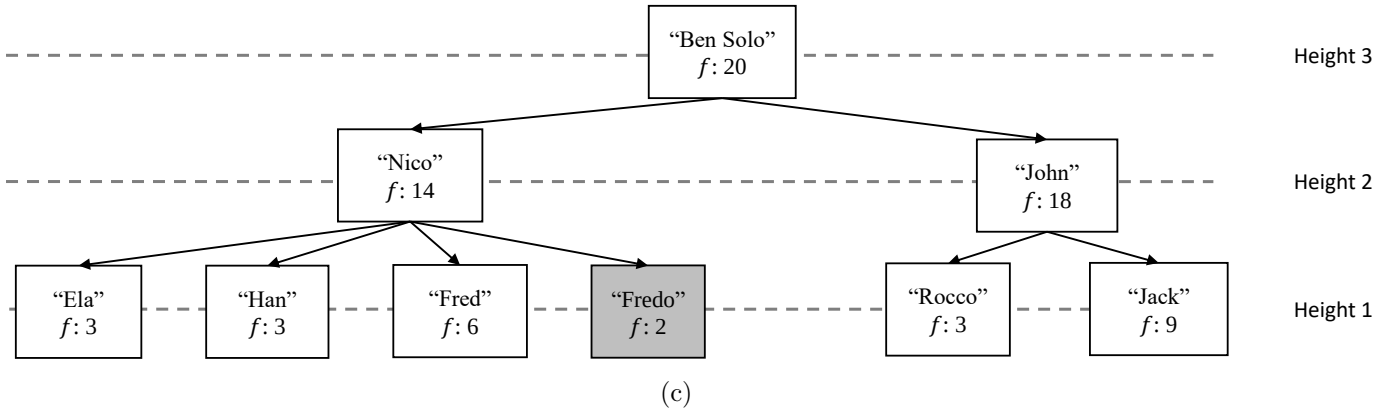
```
>>> H = ["Ben Solo", 20,  \
            ["Jack", 9,  \
               ["Fredo", 2,  ["Kyle", 1], ["Luke", 1]], \
               ["Vincenzo", 1],  \
               ["Fred", 6, ["Ela", 3], ["Han", 3]] \
            ], \
            ["John", 18, ["Rocco", 3], ["Nico", 14]] \
       ]
>>> help_ben_solo(H, 2)
(['Ben Solo', 20, ['Nico', 14, ['Ela', 3], ['Han', 3], ['Fred', 6], ['Fredo', 2]], ['John', 18,
  ['Rocco', 3], ['Jack', 9]]], ['Kyle', 'Juke', 'Vincenzo'])
```




(a)




(b)




(c)

Figure 1: (a) The First Order as Ben Solo has inherited from the previous Supreme Leader. (b) The First Order after addressing Constraint 1. The shaded boxes have been swapped. (c) The New Order after addressing Constraint 2 on the hierarchy in subfigure (b). The shaded box has been moved up in the hierarchy.

## Notes

- You may use the `dict` data type.
- You are not allowed to import any function or module.

- You may define helper functions.
- You may use recursion or iteration.
- Your function will be tested with multiple data.
- Any program that performs below 30% of the total grade will enter a glass-box test (eye inspection by the grader TA). The TA will judge an overall grade in the range of [0,30].
- The glass-box test grade is not open to negotiation, discussion or explanation.
- Your function will be tested with Python interpreter (v2.7) that is installed on *inek* machines running Linux.
- You are encouraged to share input-outputs on the newsgroup of the course.