6.009 Optional Lab: Who's got Talent?

This lab is *optional*

This lab assumes you have Python 3.5 installed on your machine.

Introduction

We are auditioning candidates for a show called "Who's got Talent?" Each candidate is capable of performing a set of talents. Some candidates might be capable of performing multiple talents (e.g., Singing, Dancing, Humor, etc.), while others might be able to perform only one or none at all. In this lab you will write an algorithm for selecting which candidates to perform on the show. Every selected candidate will perform all of his or her talents. The producer has given you two constraints.

First, the producer wants "Who's Got Talent?" to draw a wide audience -- this means incorporating a wide range of talents into the show. You will be given a set of talents that the producer cares about. Every one of these talents must be performed in the show.

Second, the producer wants to limit costs, which are proportional to the number of performers. You need to select as few performers as possible.

Thus your ultimate goal is to select the fewest possible performers, while still ensuring that every talent that the producer cares about will be performed.

Problem Setup -- with an Example

For convenience, we identify the candidates by numbering them: Candidate 0, Candidate 1, Candidate 2, etc. We number the talents as well: Talent 0, Talent 1, Talent 2, etc.

There are multiple ways to represent our universe of candidates and talents. For your convenience we provide two list-of-list representations, `candidate_to_talents` and `talent_to_candidates`. `candidate_to_talents[c]` is the list of talents that Candidate `c` is able to perform. On the other hand, `talent_to_candidates[t]` is the list of candidates who are able to perform Talent `t`. **For convenience, we assume that the producer wants to include every talent in the show.** (This is not much of an assumption, because otherwise we could define our universe to only include the talents that the producer cares about.)

For example, consider the following candidate-talent universe, where "x" denotes the ability of a candidate:

Candidate	Talent 0	Talent 1	Talent 2	Talent 3
Candidate 0	x	x	x	x
Candidate 1		x		x
Candidate 2		x	x	

In our representation, `candidate_to_talents` is the following list of lists:

```
candidate_to_talents[0] = [0, 1, 2, 3]
candidate_to_talents[1] = [1, 3]
candidate_to_talents[2] = [1, 2]
```

Similarly, `talent_to_candidates` is the following list of lists:

```
talent_to_candidates[0] = [0]
talent_to_candidates[1] = [0, 1, 2]
talent_to_candidates[2] = [0, 2]
talent_to_candidates[3] = [0, 1]
```

`lab.py`

You will modify `lab.py` to complete this lab. In that file, you will need to correctly implement the `select_candidates()` function as specificed.

Your code will be loaded into a small server (`server.py`, much the same as in the other labs). It serves up a website that might help you visualize the output of your function. The candidates selected by your algorithm will be highlighted. The UI also presents feedback when your algorithm times out or when your algorithm determines that no solutions exist.

To use the visualization, run `server.py` and use your web browser to navigate to localhost:8000 (http:://localhost:8000). You will need to restart `server.py` in order to reload your code if you make changes.

Basic Functionality

You should develop basic functionality of the `select_candidates()` function, which takes the following arguments:

- `num_candidates`: the number of candidates in our universe, where candidates are numbered from `0` to `num_candidates - 1`
- `num_talents`: the number of talents in our universe, where talents are numbered from `0` to `num_talents - 1`
- `candidate_to_talents`: list of lists, as discussed above
- `talent_to_candidates`: list of lists, as discussed above.

`select_candidates` should return a list of candidates, of the smallest possible size, that is able to perform all of the talents. If no such list of performers exists, then `select_candidates` should return an empty list. In the example above, `select_candidates` should return `[0]`, because Candidate `0` is able to perform all of the talents alone.

There are a number of ways you could go about implementing `select_candidates`. For the purposes of this lab we expect you to take a recursive approach. Perhaps the most important aspect of solving a recursive problem is framing the problem correctly -- it is well worth the time to carefully set up your approach.

Unit Tester (`./test.py`)

As in the previous labs, we provide you with a `test.py` script to help you verify the correctness of your code. We will only use the provided test cases to auto-grade your work. You may also wish to investigate the Python debugger (PDB) to help you find bugs efficiently.

Does your lab work? Do all tests in `test.py` pass? You're done!