Computing for Medicine: Phase 3, Seminar 5 Project

Rabiya Noori

Based on slides by Jennifer Campbell

Package Installation

> conda activate C4M (Windows)

> source activate C4M (Mac)

Packages needed for this project:

- requests

> conda install -c anaconda requests

Starter code and data

Starter code (in the ZIP file)

- phenotips_project.py (TODOs)
- ontology_parser.py (complete)
- ontology_explorer.py (TODOs)

Data

- hp.obo (Human phenotype ontology)

Your tasks

0) Explore PhenoTips using your web browser.

1) Write a program to interact with PhenoTips.

2) Write a program to get information about the Human Phenotype Ontology (HPO).

3) Q&A: revisiting design decisions; more exploration

Programming Concepts

set

- Python sets are unordered collections of unique immutable objects.
- https://docs.python.org/3/library/stdtypes.html#set

– s = set() # an empty set

– s.add(1)

– s.add(2)

– s2 = set([1, 2, 3, 4]) # new set with 4 items

– s2.add(3) # 3 already in s2, so s2 is unchanged

- Note: using type set is not a requirement, but you may find it helpful.

Assigning parameters default values

- For certain functions, including range and print, the number of arguments that you pass to them can vary.

For example:

– print(‘hello’)

– print(1, 2, 3)

– print(‘a’, ‘b’, ‘c’, end=’xyz')

– print(1, 2, 3, sep='..', end='!')

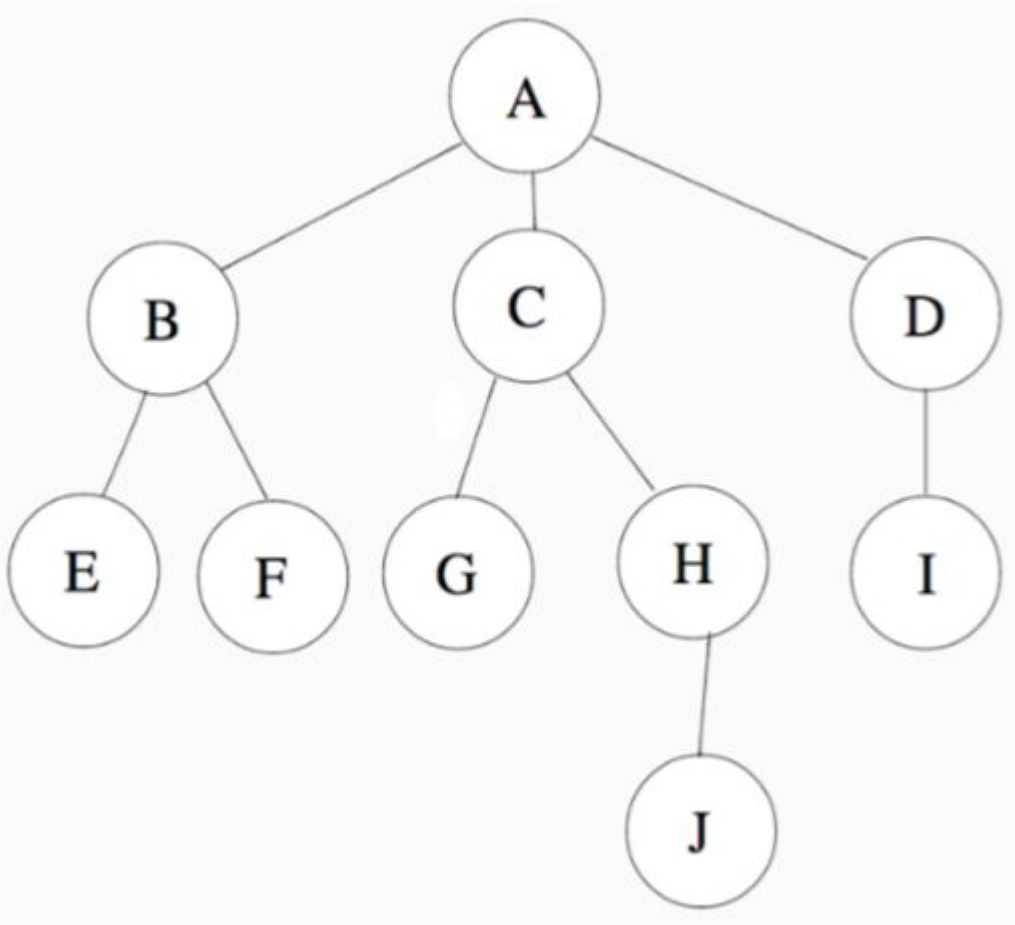
- Demo: default_parameters.py

Recursion

- To solve a problem, identify how it can be broken down into smaller instances with the same structure.
- A recursive function is a function that calls itself.
- Any problem that we can solve with recursion can be solved with iteration (loops) and vice versa.
- Some problems have simple recursive solutions and complex iterative solutions.
- Demo: search_recursive.py, reverse_recursive.py

Trees

- Trees can represent data that has a hierarchical structure.
- A, B, C ... J are nodes.
- A is the root of the tree.
- A is the parent of B, C, D.
- B, C, D are children of A.
- E, F, G, J, I are leaf nodes(nodes with no children).



More recursion: getting leaf nodes

- Demo: tree_recursive.py

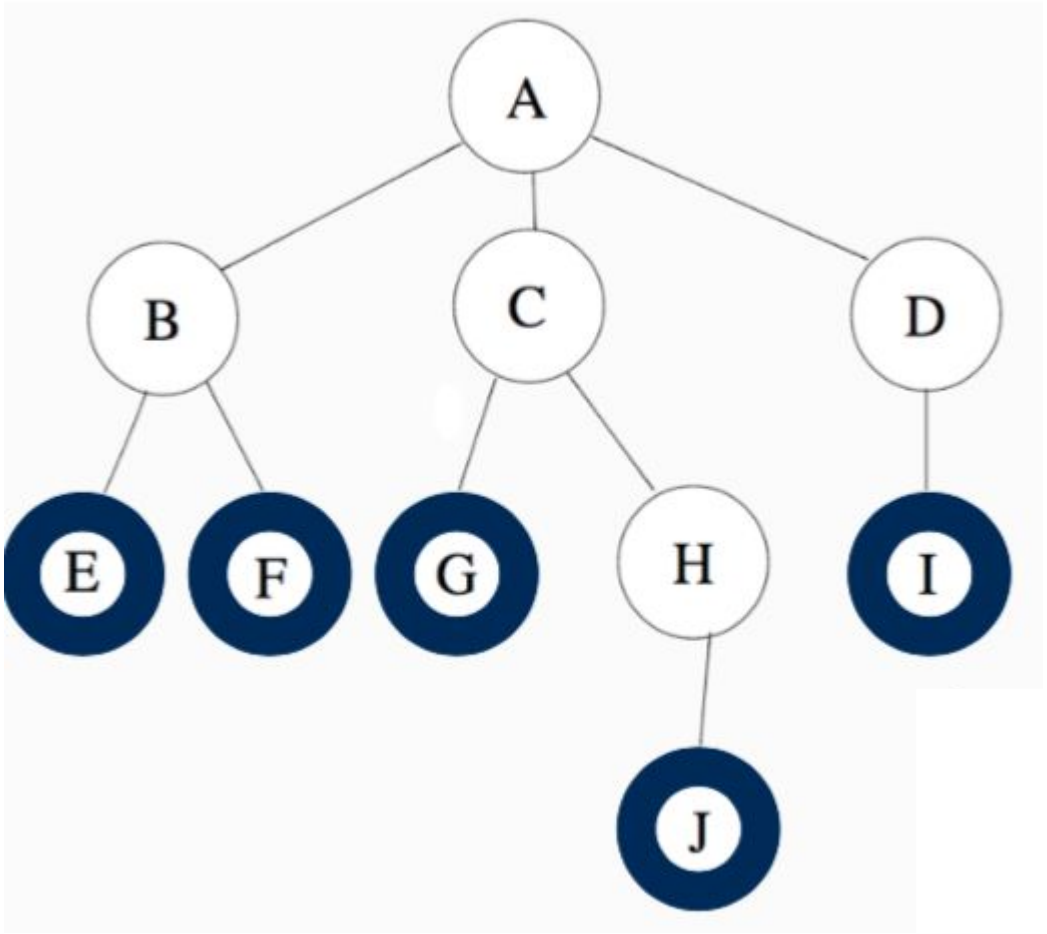
tree =
{'A': ['B', 'C', 'D'],

'B': ['E', 'F'],

'C': ['G', 'H'],

'H': ['J'],

'D': ['I']}



Starter code: class Ontology

- You already know how to use Python's classes (e.g., str and list) and methods (e.g., str.startswith and list.append).
- In ontology_parser.py , a class named Ontology is defined.
- For this project, you will use class Ontology and call on its methods.
- You do not need to know how to define classes to complete this project. However, if you would like to learn more about object-oriented programming, you can ask me.

Ontology representation

The starter code produces two dictionaries to represent the human phenotype ontology:

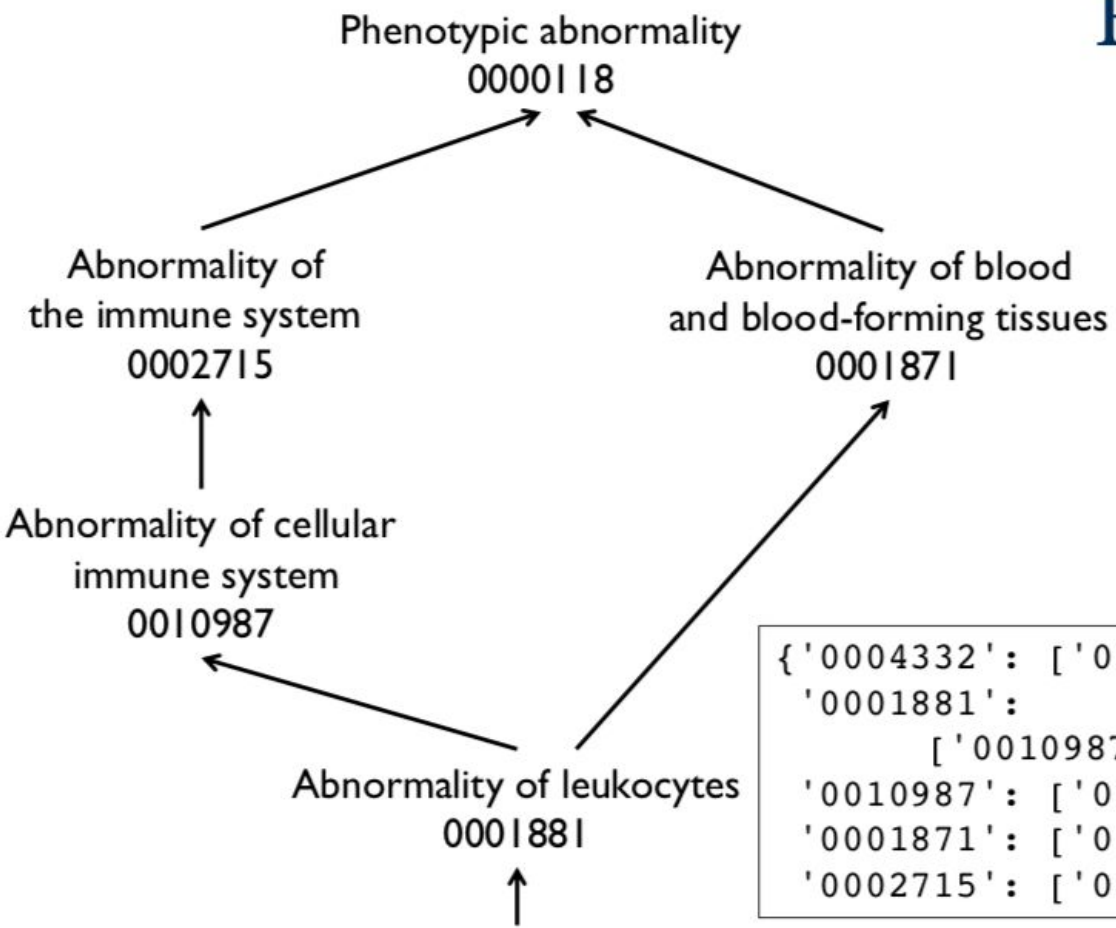
- pid_to_name: each key is an ID for a phenotypic feature and each value is its name
- pid_to_parents: each key is an ID for a phenotypic feature and each value is a list of its parent IDs

Tips

Test your code as you write it, a little bit at a time.

- The Ontology class represents the HPO using a dictionary that maps a child node to its parents, unlike the tree dictionary which mapped a parent to its children. If you prefer to have the opposite, you can write a helper function to invert the dictionary.

Example




```
{'0004332': ['0001881'],
 '0001881':
       ['0010987','0001871'],
 '0010987': ['0002715'],
 '0001871': ['0000118'],
 '0002715': ['0000118']}
```