DS 2500: Intermediate Programming with Data
Prof. Rachlin
Northeastern University

Property Graphs and Network Analysis

Problem Description:

In this assignment we will use object-oriented methods to implement a property graph model which will serve as the basis for a framework for doing network-centric data analysis. Many big-data problems are most naturally modeled as graphs, and graph databases are an important category of NoSQL (non-relational) databases, providing an alternative to traditional databases such as MySQL, SQL Server, and Oracle where all data is stored in tables. Applications of graph-based models include social networks, biological pathway modeling, routing problems, and supply-chain optimization.

Graphs are a collection of nodes (vertices) and relationships (edges). Property graphs are an extension of the basic graph model where both the nodes and the relationships can take on properties (key/value pairs).

- Nodes have a name or *label*, a type category, and an optional collection of key/value properties.

- Relationships are directional, each having a start and an end node. Like nodes, labels might also have a label, a type category, and their own collection of key:value properties. (Often the labels are blank or empty).

Property Graphs enable us to model and reason about connected knowledge. In class we used a simple dictionary to create a simple adjacency-list representation. The underlying model is a dictionary where nodes are keys and values are a simple list of adjacent nodes. Here we need something a little more complicated. Again, the nodes could be keys in a dictionary. But the values should be lists or sets of *tuples* (Node, Relationship) that identifies the relationship object attaching one node to another node. For example, if node A is connected to node B using relationship R_{AB}, and node A is connected to node C using relationship R_{AC}, then the dictionary might look like this:

{ A: [(B, R_{AB}), (C, R_{AC})] }

When adding an edge we'll assume that the edge is a directed (one-way) edge. Now when finding adjacent nodes, we *might* want to be selective based on the category or properties of the adjacent nodes and relationships involved.

Part A: Classes and Methods to implement:

Node

- **__init__**: The constructor
- **get_property(key)**: Get the value for a specified property key
- **get_all_properties()**: Return all properties for a given node

Relationship

- **__init__**: A constructor
- **get_property(key):** Get the value for a specified propery key
- **get_all_properties():** Return all properties for a given relationship

Property_Graph

- **__init__**: The constructor
- **add_node(n):** Add a node to the graph
- **add_relationship(n,m,rel, directed=True):** Add a relationship (rel) to the graph connecting node n to node m. The relationship is assumed to be directed unless directed=False in which case it is undirected.
- **find_nodes(label=None, category=None, key=None, value=None)**: Find nodes meeting certain criteria. For example, having a particular name / label or category type, or matching a specific property.
- **subgraph(nodes)**: Return the subgraph consisting of a specified set of nodes and any interconnecting edges.
- **adjacent(n, node_categogry=None, rel_category=None)**: return all adjacent nodes, possibly constrained to a particular category of nodes and/or relationships. You may optionally want to add further constraints on node / relationship properties.
- **visualize()**: Render the graph visually. Bonus points for supporting color maps on node categories.

- ***optional*: shortest_path(n,m)**: Find the shortest path from a *start* node to an *end* node using breadth-first search. Return a subgraph consisting of the shortest path. (We implemented this in class.)

  ***optional*: degree_distribution()**: Plot the degree distribution of the network. (We implemented this in class.)

Part B: Putting the Property Graph class to work

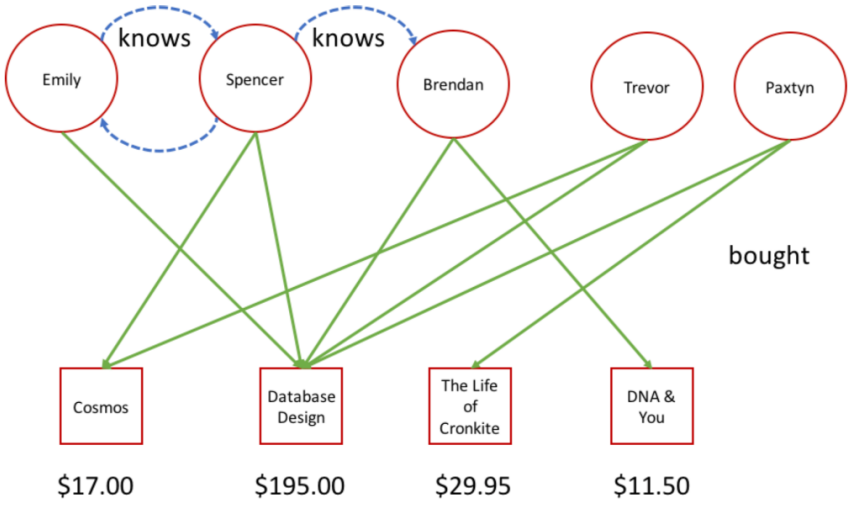
We'll experiment with our property graph model by applying it to two different applications: book recommendations and gene/disease associations.

Book Recommendations

Build the graph model diagrammed below. This can be hard-coded.

In this model, the node label is the name of a person or the title of a book, the category is "Person" or "Book". The books have an additional price property. Relationships are unnamed and have no properties. The category of the relationship is either "Knows" or "Bought".

Use your property graph methods to find all books bought by people that Spencer knows. This will be the basis of our book-recommendation engine. Implement **__repl__** and/or **__str__** on the node in order to output the recommendations.



Gene/Disease associations

Read the gad data provided into a property graph. The node categories are "Gene" and "Disease". The gene column is the name of the gene and the disease column contains the name

of the disease. Chromosome is a gene property. Disease_class is a disease property. The relationship category will always be “associated_with”. The three columns, num_positive, num_negative, num_unknown, are relationship properties denoting the number of publications that report a positive, negative, or unknown link between the gene and the disease.

Use our property_graph model to:

a) Find all genes positively linked to asthma
b) Find all diseases positively linked to any of these asthma genes
c) Visualize the subnetwork representing asthma-linked genes and the diseases associated with these genes.

What to submit:

Submit your code and analysis for grading. You may make simplifications to the assignment as needed for partial credit. It is better to submit an incomplete assignment than no assignment at all.