DS 4300: Large Scale Storage and Retrieval
Prof. Rachlin
Northeastern University

Property Graphs and Network Analysis: A scala-based in-memory graph database

Problem Description:

In this assignment you will implement your own, home-grown, PropertyGraph model using Scala. You will then use the PropertyGraph to instantiate some data and demonstrate support for various network-centric tasks. The implementation will be similar to the WeightedGraph class we implemented in class. Recall for our implementation of a basic unweighted Graph is an adjacency list implemented using a Map[Node,Set[Node]] object where the keys are Node objects and the values are a set of Nodes. For the WeightedGraph we extended this so that the implementation is now a Map[Node,Set[(Node,Double)]]. In other words, the key is (still) a Node and the value is now a set of (Node, Double) tuples. Here, the double corresponds to the weight of the associated edge. For example, if node A is connected to node B via a weighted edge of weight 5, while connected to C with weight 3, our Map would include:
A → Set((B, 5.0), (C, 3.0)) and so on.

In property graphs, Node objects have a name, a category, and a Map[String, Any] containing additional properties. We implemented the Node class recently in lecture. In addition, nodes are connected via *Relationships* (edges). A relationship doesn't have a name, but in our model, it will have a category and a collection of properties, just like a Node. The implementation of PropertyGraph is quite similar to WeightedGraph, except now instead of the implementation being a Map[Node,Set[(Node,Double)]] (mapping Nodes to sets of [Node,Double] tuples) it is now a Map[Node,Set[(Node,Relationship)]] (mapping Nodes to sets of [Node,*Relationship*] tuples.

Part A. Implement classes for Node, Relationship, and PropertyGraph.

Your first task is to modify WeightedGraph (shown in class) to make it a PropertyGraph. Simple weighted edges are being replaced by Relationship objects. You will need to implement three classes: Node, Relationship, PropertyGraph.

Methods on Node

// Add a property to a node
def addProperty(key: String, value: Any): Unit

// Get a property from a node
def getProperty(key: String): Any

// Return the node as a string representation
override def toString: String

Methods on Relationship

// Add a property to a relationship
def addProperty(key: String, value: Any): Unit

// Fetch a property from a relationship
def getProperty(key: String): Any

// Returns a string representation of a Relationship object
override def toString: String

Methods on PropertyGraph

// Add a node to the property graph
def addNode(n: Node): Unit

// Add a relationship to the property graph
def addRelationship(n: Node, m: Node, r: Relationship): Unit

// Find nodes that match a name and/or a category
// If the name isn’t given, ignore the name constraint
// If the category isn’t given, ignore the category constraint
def match(name: String = “”, category: String = “”): Set[Node]

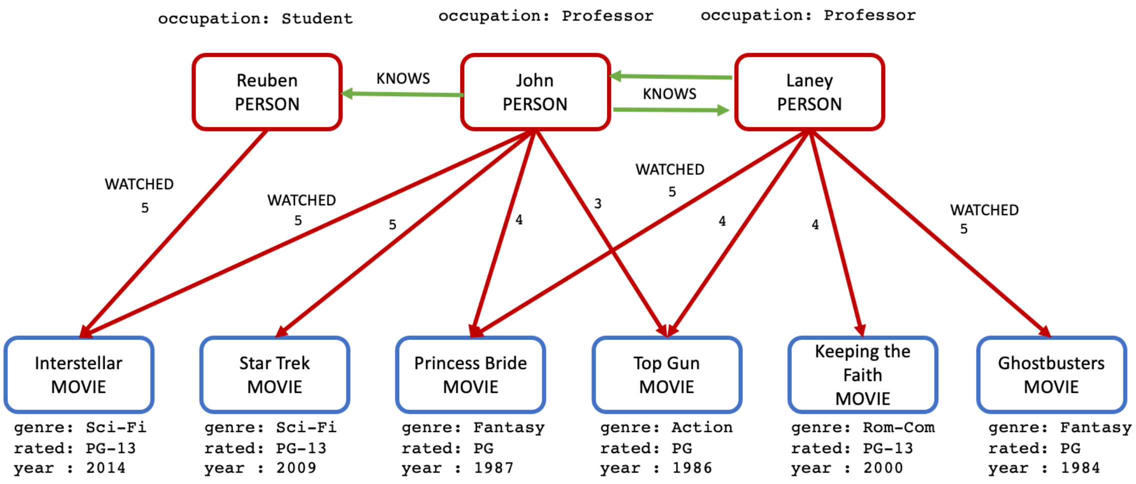
// Find nodes adjacent to a given node
// Optionally restrict the results to nodes matching a node category
// and/or nodes connected via relationships involving a particular relationship category
def adjacent(n: Node, nodeCategory: String = “”, relCategory: String = “”): Set[Node]

// Find the subgraph consisting of the listed nodes and any relationships that occur
// between any pair of nodes in the list.
// Return a *new* PropertyGraph object.
def subGraph(nodes: List[Node]): PropertyGraph

```
// String representation of the graph
// Output something that is human-readable and that documents all of the nodes
// and relationships and their properties
override def toString: String
```

Part B: Instantiate a property graph with some data and print the graph

Here is a picture of an example property graph about people and the movies that they watched and how they rated those movies. All the red edges are of category = “WATCHED” (some of the labels are missing in the diagram). The number is a *rating* property (how the user rated the movie on a scale of 1 to 5).



Part C: Network Analysis using Property Graphs

1. Extract and print the sub-graph consisting of just the PEOPLE nodes.

2. Extract and print the sub-graph consisting of John and the 4 movies that he watched.

3. Let’s use our PropertyGraph as a recommendation engine. Find all the movies that were watched by people that John knows. Discard any movies that John has already watched. Print the set of recommendations.

What to submit:

Submit your code and any outputs for grading.