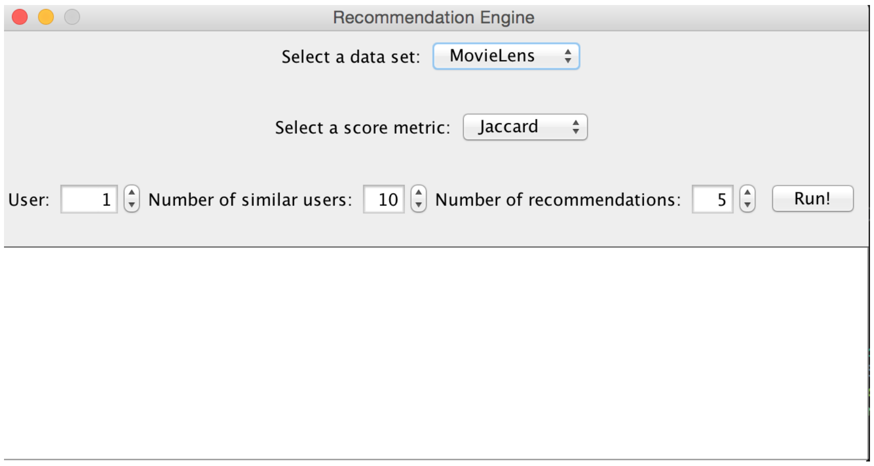
Nets 150: Homework , Part 2 - User Manual

Trevin Gandhi, Deepan Saravanan, Seth Bartynski

Using the GUI Application

1. To run the GUI for our recommendation engine, run GUI.java as a java application

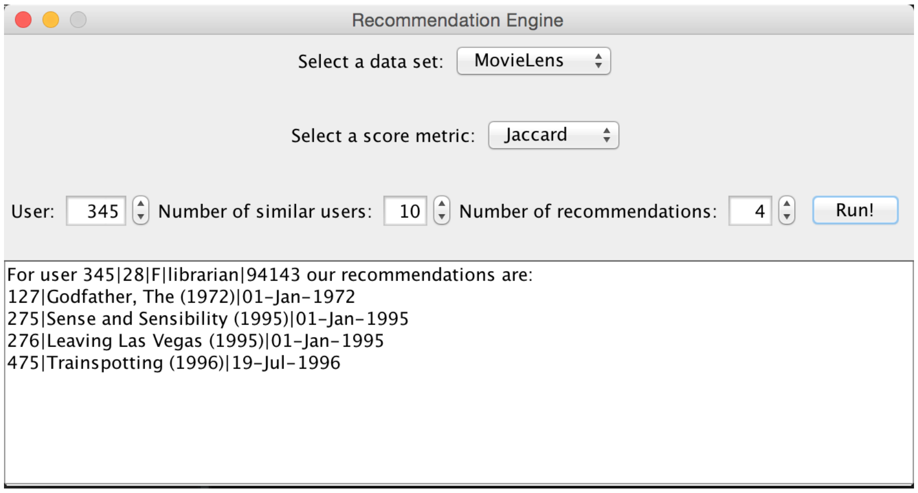
2. When run, a window similar to the one pictured below should appear on the screen



3. Through the GUI Application, the user is able to select a scoring metric (Jaccard or Pearson) through which the data is analyzed

4. After selecting a scoring method, the user may then find recommendations by adjusting the three parameters given below the scoring metric and clicking run

5. After running, GUI Application will provide recommendations similar to the picture displayed below



```
For user 345|28|F|librarian|94143 our recommendations are:
127|Godfather, The (1972)|01-Jan-1972
275|Sense and Sensibility (1995)|01-Jan-1995
276|Leaving Las Vegas (1995)|01-Jan-1995
475|Trainspotting (1996)|19-Jul-1996
```

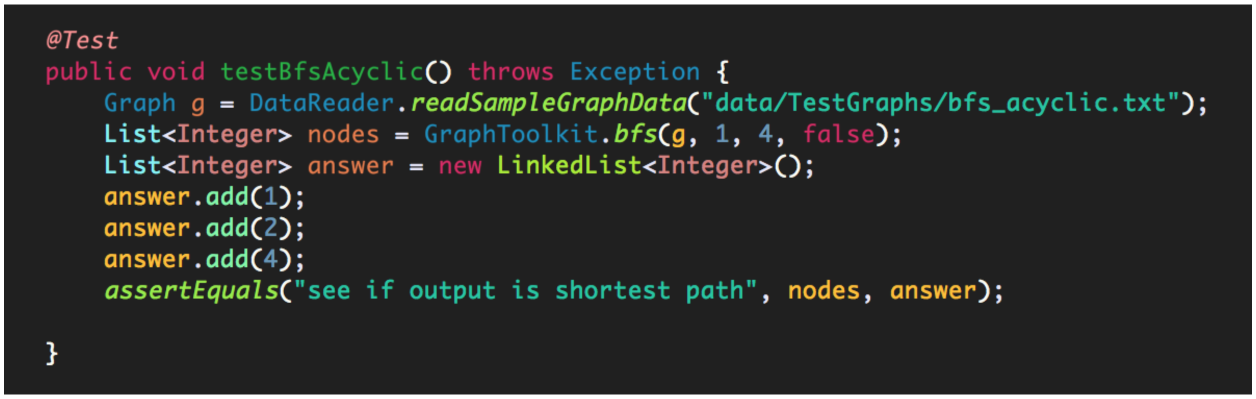
Using the Graph Toolkit

1. The Graph Toolkit is built to accept and return user IDs, which are then translated into Nodes that are private to the engine package. Thus, outside users are expected to interact with the API through user IDs, which are available through the source data files.

2. Every graph algorithm contained in GraphToolkit.java has multiple test cases in GraphToolkitTest.java which test for the correct behavior. Algorithms in GraphToolkit.java can be used the same way as the test use the algorithms.

3. The graph algorithm tests in GraphToolkitTest.java call upon the implementation of the algorithm in GraphToolkit.java for various inputs.

4. An example of using the GraphToolkit API with user id inputs is given below in the following test case.



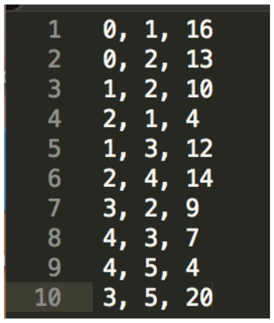
```java
@Test
public void testBfsAcyclic() throws Exception {
    Graph g = DataReader.readSampleGraphData("data/TestGraphs/bfs_acyclic.txt");
    List<Integer> nodes = GraphToolkit.bfs(g, 1, 4, false);
    List<Integer> answer = new LinkedList<Integer>();
    answer.add(1);
    answer.add(2);
    answer.add(4);
    assertEquals("see if output is shortest path", nodes, answer);

}
```

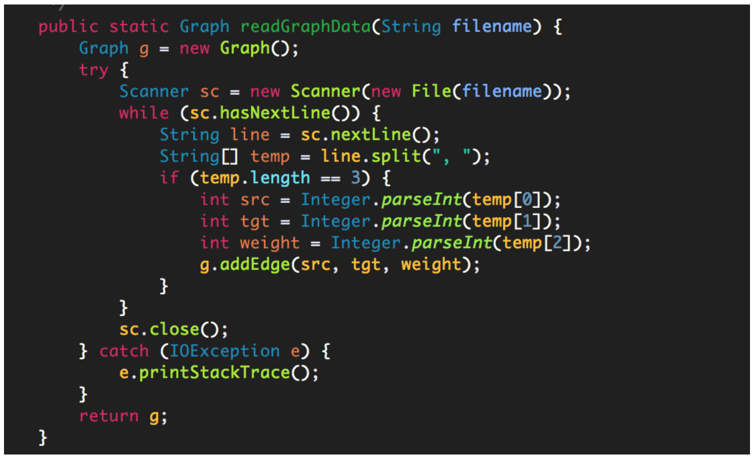
Using the Data Reader

1. DataReader.java provides the implementation for reading from text files and storing data as a graph. The graph representation is then used in the GraphToolkit API and in generating recommendations.

2. Reading Data for GraphToolkit API
    a. The text file read by DataReader.java must be formatted in a certain format. Specifically the text file must contain list of 3-tuples. An example of a valid text file is provided below. In this representation, any given 3-tuple, say (x, y, z), represents a graph such that x and y detail an edge between nodes x and node y, and z represents the flow on the edge between x and y. Provide below is also code from the DataReader class that constructs a graph from the given text files.

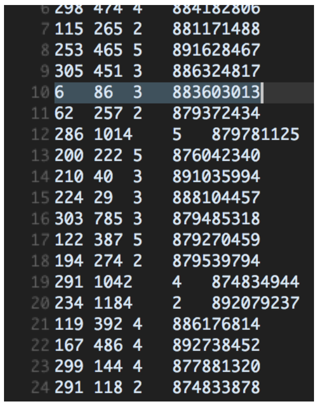


```
0, 1, 16
0, 2, 13
1, 2, 10
2, 1, 4
1, 3, 12
2, 4, 14
3, 2, 9
4, 3, 7
4, 5, 4
3, 5, 20
```

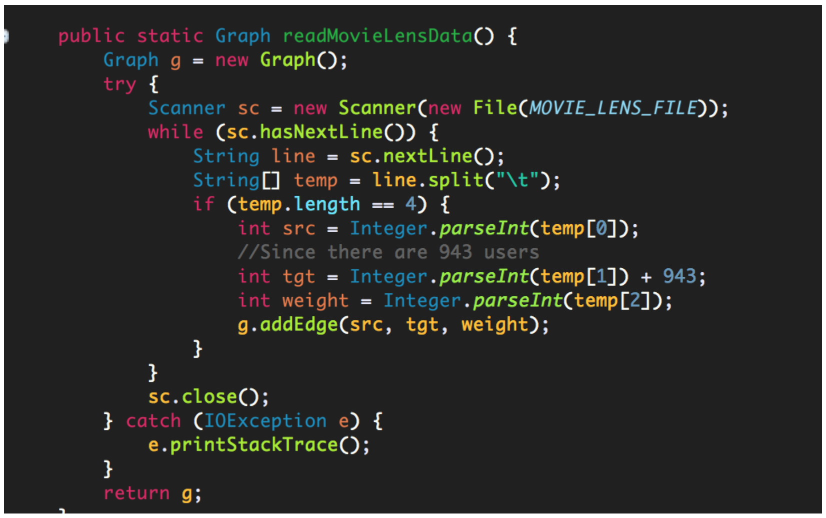


```java
public static Graph readGraphData(String filename) {
    Graph g = new Graph();
    try {
        Scanner sc = new Scanner(new File(filename));
        while (sc.hasNextLine()) {
            String line = sc.nextLine();
            String[] temp = line.split(", ");
            if (temp.length == 3) {
                int src = Integer.parseInt(temp[0]);
                int tgt = Integer.parseInt(temp[1]);
                int weight = Integer.parseInt(temp[2]);
                g.addEdge(src, tgt, weight);
            }
        }
        sc.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return g;
}
```

3. Reading Data for MovieLens Datasets
    a. The MovieLens Dataset is represented as a list of 4-tuples. An snippet of the data from the MovieLens Dataset is provided below. An example of a valid text file is provided below. In this representation, any given 4-tuple (x, y, z, w), x represents a user id, y represents a movie id, z represents a rating, and w represents a timestamp (which we disregard when reading in the data).



```
298 474 4   884182806
115 265 2   881171488
253 465 5   891628467
305 451 3   886324817
6   86  3   883603013
62  257 2   879372434
286 1014    5   879781125
200 222 5   876042340
210 40  3   891035994
224 29  3   888104457
303 785 3   879485318
122 387 5   879270459
194 274 2   879539794
291 1042    4   874834944
234 1184    2   892079237
119 392 4   886176814
167 486 4   892738452
299 144 4   877881320
291 118 2   874833878
```



```java
public static Graph readMovieLensData() {
    Graph g = new Graph();
    try {
        Scanner sc = new Scanner(new File(MOVIE_LENS_FILE));
        while (sc.hasNextLine()) {
            String line = sc.nextLine();
            String[] temp = line.split("\t");
            if (temp.length == 4) {
                int src = Integer.parseInt(temp[0]);
                //Since there are 943 users
                int tgt = Integer.parseInt(temp[1]) + 943;
                int weight = Integer.parseInt(temp[2]);
                g.addEdge(src, tgt, weight);
            }
        }
        sc.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return g;
```