CS6770 - Knowledge Representation & Reasoning Implementation of FCA Algorithm to build the concept lattice

(CS13M001 - AASHIMA BHATIA, CS13M006 - ARJUNLAL B)

April 21, 2014

Problem Statement :

Given a context in the form of a table implement the FCA algorithm to build the concept lattice. Display the lattice graphically. Given a concept C one should be able to view the extent and the intent. Extension: Given a numeric attribute a user should be able to select ranges with memberships to these ranges as attributes.

Status of the assignment :

Complete

Language Used :

JAVA has been chosen as the language for implementation.

Approach Used :

Following the discussions with Prof. Deepak Khemani, the Galois lattice construction algorithm (http://arxiv.org/pdf/cs/0602069.pdf) proposed by Vicky Choi (2006) has been implemented.

Details :

Input : The program takes the concept table as input in the form of a formatted text file, where the first line is the number of objects in the file and the rest of the file contains concept table. A sample input file is shown below

```
4
BOOL|    Prop1          Prop2          Prop3          Prop4
Obj1     1              1              0              1
Obj2     0              1              0              0
Obj3     1              1              0              0
Obj4     0              0              1              1
```

Data Structures used : The program constructs the data structures needed for the algorithm using this input file and work on them as the algorithm specifies. Various data structures such as sets, lists, arrays and class objects have been used.

Output : The program outputs two files. The file 'output.gv' contains the lattice, output in DOT language (graph description language). The nodes in the graph is represented using their node ID number. The second file - 'index.txt', gives an index, mapping each node ID to its corresponding concept. Sample output is given below.

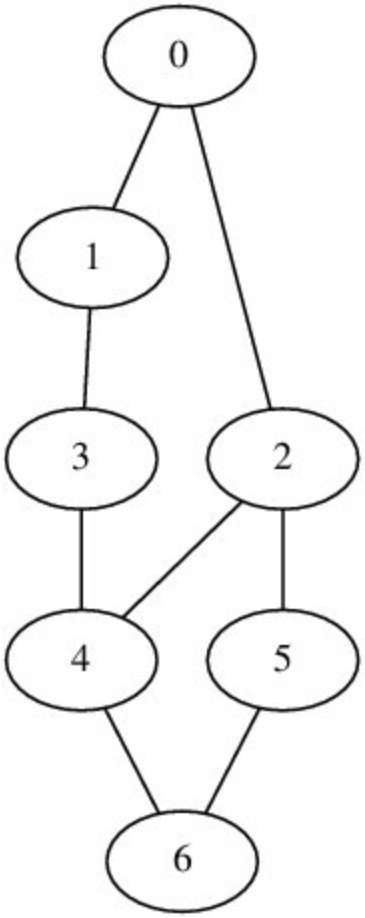
output.gv

```
graph fca {
0--1;
1--3;
3--4;
0--2;
2--4;
4--6;
2--5;
5--6;
}
```

index.txt

```
0--[][Obj1, Obj2, Obj3, Obj4]
1--[Prop2][Obj1, Obj2, Obj3]
3--[Prop1][Obj1, Obj3]
4--[Prop2, Prop1, Prop4][Obj1]
2--[Prop4][Obj1, Obj4]
4--[Prop2, Prop1, Prop4][Obj1]
6--[Prop2, Prop1, Prop3][]
5--[Prop3][Obj4]
6--[Prop2, Prop1, Prop3][]
```

The graph representation of 'output.gv' file is given below



Extension :

The mentioned extension also has been implemented. If there are numerical values that ties a property to an object, the program is capable of taking an extra file which specifies the range of values for the corresponding property and then converts the value

into boolean based on whether the given number falls within the specified range or not. A sample file specifying the range of values is shown below

```
Prop1   10      100
Prop2   200     1500
```

Syntax for file specifying numerical ranges for property is

```
property1Name minValue maxValue
property2Name minValue maxValue
...
```

Execution of the program :

Program can be executed by using the command

```
java fca inputFileName
```

If the properties are numerical and the range specification file is also to be inputted, execute it as follows

```
java fca inputFileName rangeSpecFileName
```

Execution of the example from textbook :

Input

```
6
Objects Maths   English Science games   Music   History
Avinash 0       0       1       0       0       0
Arnav   0       1       1       0       0       0
Aarti   1       1       1       1       1       0
Amrita  0       0       1       1       0       0
Ashwani 0       0       1       0       0       0
Ashok   0       0       0       0       0       0
```

output - index file

```
0--[][Amrita, Avinash, Ashwani, Aarti, Ashok, Arnav]
1--[Science][Amrita, Avinash, Ashwani, Aarti, Arnav]
2--[games][Amrita, Aarti]
4--[Maths, games, Music, English][Aarti]
5--[Maths, games, History, Music, Science, English][]
3--[English][Aarti, Arnav]
4--[Maths, games, Music, English][Aarti]
```

output - graph