**Project 3**

**Implementing Algorithms on Functional Dependencies**

1. **Finding Closure of a set of attributes.**  
     
   Attribute closure of an attribute set can be defined as set of attributes which can be functionally determined from it.  
   Given Functional Dependency A -> B, then Attribute Closure of A will have A and B.  
   /\*\*

\* Find the closure of attribute set attr under functional dependencies fds.

\* Refer to Algorithm 16.1 (15.1 in edition 7) for more details

\*

\* @param attr, list of attributes. Each attribute is an uppercase letter "A", "B", etc

\* @param fds, list of functional dependencies of format "AB->XY"

\* @return closure of attributes attr under fds

\*/

public List<String> findClosure(List<String> attr, List<String> fds);

1. **Finding Minimum Cover**We say that a set of functional dependencies F covers another set of functional dependencies G, if every functional dependency in G can be inferred from F. More formally, F covers G if closure of G ⊆ closure of F. F is a minimal cover of G if F is the smallest set of functional dependencies that cover G.   
   /\*\*

\* Find the minimum cover of a set of functional dependencies fds

\* Refer to Algorithm 16.2 (15.2 in edition 7)

\*

\* @param fds, list of functional dependencies of format "AB->XY"

\* @return minimum cover over set fds

\*/

public List<String> findMinCover(List<String> fds) {

1. **Finding key**The aim is to find the smallest set of attributes that can cover the entire relation R under a set of functional dependencies. This set forms the key for the relation. So given a relation R and a set of functional dependencies F, you have to find the key.  
   /\*\*

\* Find the key of relation R defined by set of attribute set attr.

\* Refer to Algorithm 16.2(a) (15.2(a) in edition 7) for more details

\*

\* @param attr, complete list of attributes in relation R. Each attribute is an  
 \* uppercase letter "A", "B", etc

\* @param fds, list of functional dependencies of format "AB->XY"

\* @return list of attributes that defines the key of relation R

\*/

public List<String> findKey(List<String> attr, List<String> fds);

1. **Converting to 3NF form**  
     
   Given a relation R and a set of functional dependencies F on the attributes of R, you can synthesize it into 3NF with dependency preservation and nonadditive join property.  
     
   This will use the algorithms implemented above.  
   /\*\*

\* Synthesize the relation into 3NF with Dependency Preservation and  
 \* Nonadditive Join Property.

\* Refer to Algorithm 16.6 (15.4 in edition 7) for more details

\*

\* @param attr, complete list of attributes in relation R. Each attribute is an  
 \* uppercase letter "A", "B", etc

\* @param fds, list of functional dependencies of format "AB->XY"

\* @return List of relations, where each relation is a list of attributes.

\*/

public List<List<String>> get3NFForm(List<String> attr, List<String> fds);

**Implementation Details:**

You have to write the above methods in FuncDep.java. There is a main class Project3.java that will invoke these methods for different test cases. The initial sample test cases are given in Input1.txt (for Closure), Input2.txt (Minimal Cover), Input3.txt (for Key) and Input4.txt (for get3NFForm).

The expected output is in ExpectedOutput.txt.

You may modify these input files to test your own cases and scenarios.

**NOTE:** You only need to turn in FuncDep.java. You do not have to turn in any other file. Do not put any logic or any changes in Project3.java or any other file. Submit this file on Blackboard.