

**Team**: Green Bay

**Professor**: [Kristin Tufte](https://trec.pdx.edu/research/researcher/Tufte/5273)

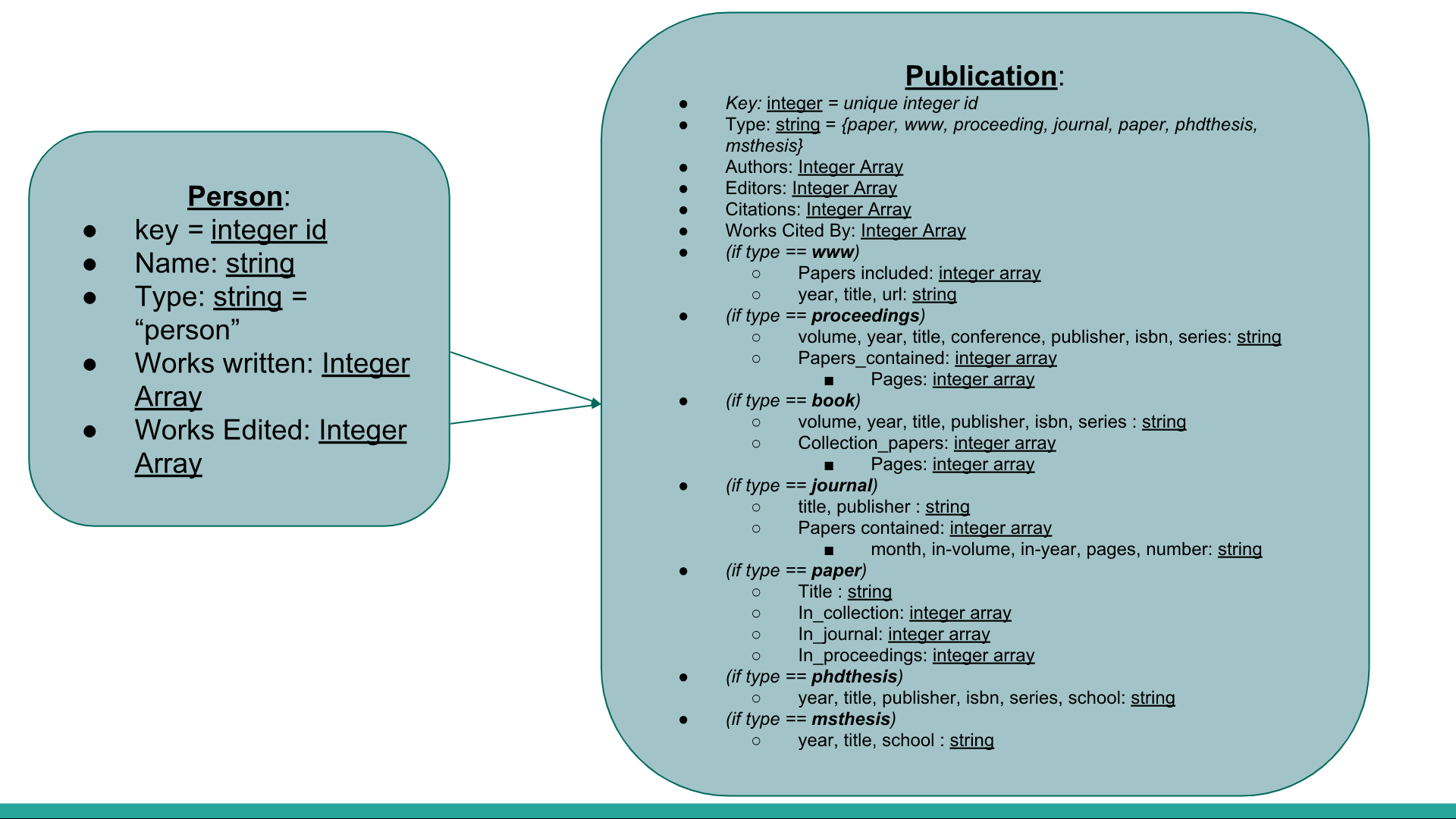
**Course**: Cloud and Cluster Data Management

**Project Part 1**: Data Modeling

**Date**: October 23, 2018

**Members:** Kevin Jacobson, Tim Pugh, Elijah Rich-Wimmer, Jason Larson Jr, Scot Lambert

# **Task 1**: *Map the given data to the key-value model, document model, or column-family model (conceptually)*



### Json Example:

|  |
| --- |
| **Person Object Example**  {  key : 1000  Name: “Name”  Type : “person”  Works written: [1004,1002,1003]  Works Edited: [1002, 1003, 1005]  } |

## 

|  |
| --- |
| **Publishing Example**  {  Key:12002  Type: “journal”  Works Cited By: [12003, 12004]  title : “The Title Data”  publisher : “The Publisher Data (NOT PERSON)”  Papers contained: 12005  } |

## 

# **Task 2**: *Describe in words or pseudo-code how you could answer the given queries.*

## 1. Co-author count: *Which publication has the most co-authors? Give full information about the paper, including title, authors and venue.*

* Algorithm Description: In this algorithm we loop over every publication object. For each object, we count the unique author ids this publication contains. We keep the key of the publication with the largest coauthor count as we iterate over the entire list. The publication key we retain at the end of this algorithm is the object with the most co-authors. We use this key to retrieve all relevant data. (Which will be known by retrieving the document variable: **type**)
* Pseudo Code:

LargestAuthor = NULL

LargestAuthorCount = -1

For each Pub.key in Publication DataStructure

localAuthorCount

For each A in Pub.Authors

#This expression will likely be simpler if we parse the string

localAuthorCount += 1

If localAuthorCount > LargestAuthorCount

LargestAuthor = Pub.key

#Because we can access the Publisher.type, we may determine every relevant data piece at this get function

Return get(LargestAuthor)

## 2. Recursive co-authors: *How many Level 3 co-authors does Michael Stonebraker have? How many does David DeWitt have?*

Algorithm description:

* For this algorithm we must find the number of level 3 co-authors provided for Michael Stonebraker and also David Dewitt. In other words, provided Stonebraker is considered a level 1 co-author (a direct author), having found his level 2 co-authors who aren’t direct level 1 co-authors (in this example, Stonebraker), find the co-authors of the level 2 co-authors who aren’t direct level 1 or level 2 co-authors (which will be Stonebraker's level 3 co-authors).
* First, map the person data structure to generate a list of works written by the author you are interested in (Michael Stonebraker and David DeWitt)
* Using each of those works written lists as the map side input to another map effort, generate a list of all other authors that are listed for those works.
* Pseudo code:

function getLvl3(person, level=1, pub=0)

Print “Searching…”

If person is empty:

print “No person selected!”

Return 0

Else if person.worksWritten[pub] is none :

Print Person.name + “doesn’t have any publications left!”

Return 0

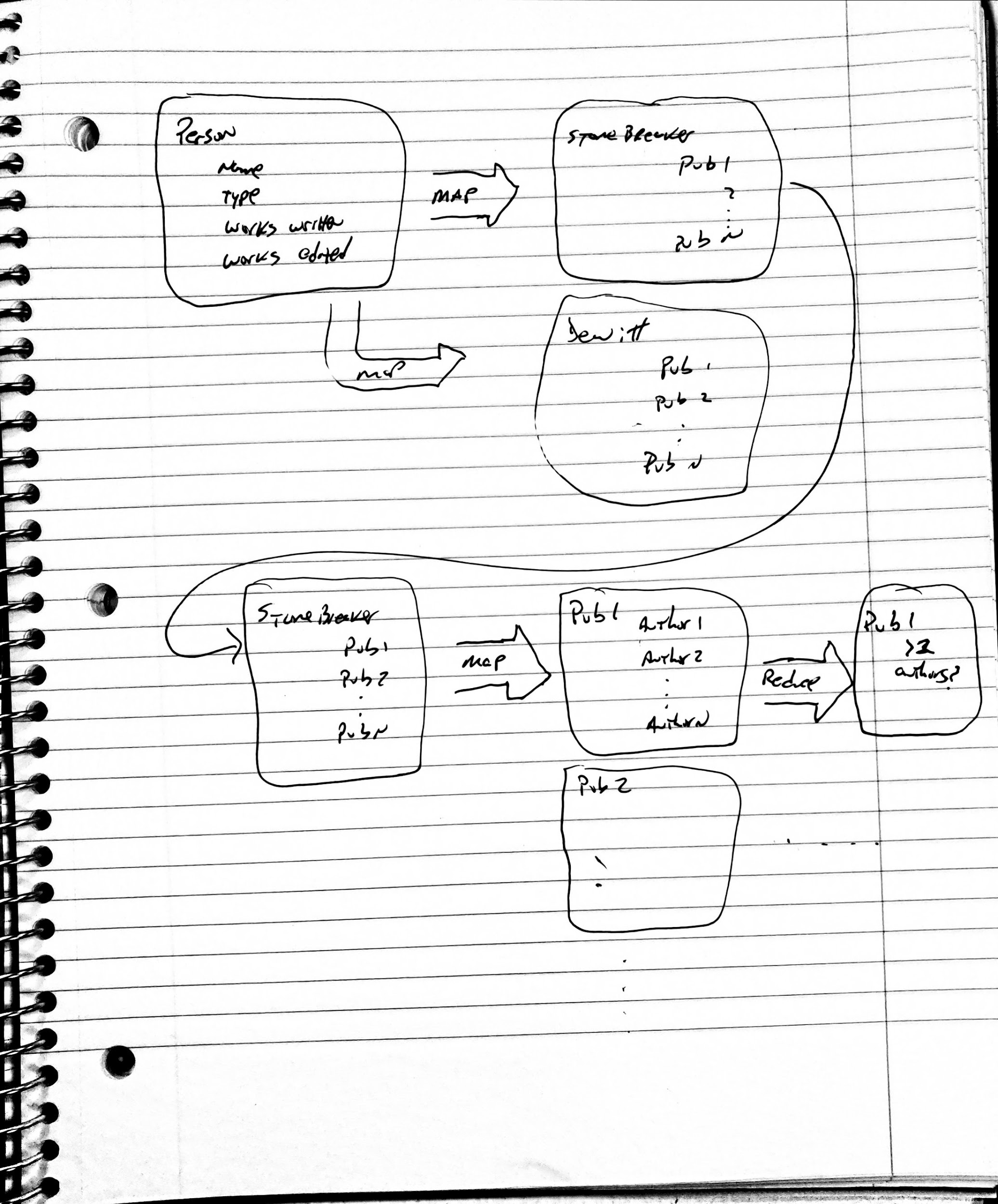
Else if level == 3:

Return lengthOfArray(Person.publications.authors) +

getLvl3(person.originalAuth(), level = 1, pub+1)

Else:

Return getLvl3(person.publication.authors.next(), level+1, pub)



## 3. Co-author distance: *At what level is Moshe Vardi from Michael J. Franklin?*

**Algorithm Description:** We first find the person objects for Moshe Vardi and Michael J. Franklin. We first check if they share any common works written that would cause them to be level 1 co-authors, if they are level 1 co-authors return level 1. If this is not the case we have to get all of the publication objects that Moshe Vardi has written. Since we know that Vardi and Franklin are not level 1 co-authors we know that Franklin’s id will not be in any of the publications. Therefore, we pull all people objects from each of the publication objects author list, excluding Moshe. We can do the same comparison between Franklin’s works written list and each of Moshe’s level 1 co-authors. If there is a match, then Franklin and Moshe are level 2 co-authors. If there is still not a match, then we repeat the search on all of Moshe’s level 2 co-authors. If a match is found then Moshe and Franklin are level 3 co-authors.

**Pseudo-Code:**

***What level is person x from person y?***

Level\_1\_co\_authors = empty

# Check if y is a level 1 co-author of x

For work in x.works\_written:

If work in y.works\_written:

Return level\_1

# Check is y is a co-author to level 1 co-authors of x

For w in x.works\_written:

For level\_1\_co in w.authors:

# Keep a list of level 1 co authors

Level\_1\_co\_authors += level\_1\_co

For work in level\_1\_co.works\_written:

If work in y.works\_written:

Return level\_2

# Check if y is a co-author to level 2 co authors of x

# for every level 1 co-author to x

For level\_1\_co\_auth in Level\_1\_co\_authors:

# for every publication by level 1 co-authors of x

For work in level\_1\_co\_auth.works\_written:

# For every co author of those works

For level\_2\_co\_auth in work.authors:

# For each publication by level 2 co-authors of x

For w in level\_2\_co\_auth.works\_written:

If w in y.works\_written:

Return level\_3

## 4. Most authors: *Which proceedings in 2010 had the most distinct authors across all papers?*

* Algorithm Description: In this algorithm, we loop over every publication object. *(If we have the ability of using a proceedings view, we may iterate over it instead.)* This algorithm returns the key of the publication object that returns the largest integer from the following query. For every publication object with the **type** variable set to *‘proceeding’*, and the **date** variable set to ‘*2010’*. For any of these objects, we collect and parse the **papers included** variable for publishing keys. For every publishing unique key, we collect and parse the **authors** variable for author keys. We count all unique keys collected from each of these queries.
* Pseudo Code:

LargestProceeding = NULL

LargestProceedingACount = -1

For each Pub.key in the Publication data structure:

Set AuthorCount to zero

Set AuthorList to empty

If Pub.count == ‘proceeding’ && Pub.year == 2010

For each Pap.key in Pub.papers\_included

Author.addeach(Pap.author)

AuthorCount = AuthorList.size

If AuthorCount > LargestProceedingACount

LargestProceeding = Pub.key

Return LargestProceeding

## 5. Triangles: *Which author participates in the most triangles?*

Algorithm Description: We loop over each person in the person data structure and look up each work they have written. For each of the works they have written, now look up the authors and ignore any entries that are the person currently under test. Store each of those as a tuple of the publication’s key and the corresponding author in a candidate set. This candidate set contains the two potential edges for our triangle for this person. Now loop over each entry in the candidate set, comparing it against all other entries in the candidate set. We are looking for two authors that co-authored another unique paper, the last edge in our triangle. If the two tuples currently being compared do not have the same publication key and the authors listed are not the same person, this is a potential combination of tuples that could yield the 3rd edge. Look up both author’s works written and take the intersection of their works written keys into a set (T), this means they were both co-authors on some third paper. Next take the set different of T with the two works written from the two tuples being compared. This will remove those works from the set and if there remains any publication key that means we have found a triangle with distinct values for each node and edge! We add that to the current count for the loop. Finally after we have checked all tuples against each other, check the count of triangles for this person against the person who we are currently saying has the most triangles. If our latest person’s count is higher, store them along with their count of triangles as the person who currently has the most.

Pseudo Code:

Store in HC, the ( Person.key, and count of triangles) with the currently known highest number of triangles.

Set CurrentCount (CC) to zero

For each P.key in the Person data structure:

Set CC to zero

#Gather up potential edges of triangle from point person

For each entry D in WorksWritten:

#we must have more than one author

Check that D.Author count is greater 1

For A each D.Author

#Don’t add the author we are currently counting for

If P.key is not equal to D.person.key:

#potential edges for our triangle

Store (D.publication.key , D.person.key) \

in a Candidate Set (CS)

#look for a potential third edge for our triangle

For each C in CS:

#We only want to check each candidate once to avoid duplicate counts.

Remove C from CS

For each E in remaining CS:

#don’t look at other authors from the same paper under test.

If E.publication.key is not equal to C.publication,

#could be different papers but the same co-author,

#ensure they are distinct authors

And E.person.key is not equal to C.person.key:

Store E.person.WorksWritten Intersection \

C.person.WorksWritten in set T

Set difference of T with (E.publication.key union C.publication.key)

If T is not empty

#we have a triangle!

Increase count of CC by 1

#check if this person has more triangles than the currently known highest.

If CC is greater than HC(count of triangles):

Store (p.key, CC) in HC

#by the end of this we should know who participates in the most triangles

Print out HC

## 6. Graph Connectivity: *Connectivity: Is the DBPL graph connected? (That is, is there a path between any two objects?)*

* **Note:** The designers of this algorithm assumed that the graph we are testing is NOT directed. Otherwise, the case of a connected graph is nearly impossible. If we were to assume a directed graph, a paper could never be considered connected to any of its authors (preventing the authors from being considered connected)



By our definition, the above graph is connected.

* Algorithm Description: The simplest algorithm to implement is a breadth first traversal of the graph. If every node within the graph is touched, then the graph is connected. If not, then there is a orphaned node somewhere in the data structure.
* Pseudo Code:

**Visited** = NULL

**Visiting** = NULL

**Current** = first key from Person Data Structure (Theoretically, this selection will be arbitrary.

do

{

**ToAdd** = empty vector

If current.type == ‘person’

**ToAdd**.add(**current**.works\_written)

**ToAdd.**add(**current**.works\_edited)

If current.type == ‘proceeding’

**ToAdd.**add(**current**.papers\_included)

If current.type == ‘journal’

**ToAdd.**add(**current**.papers\_contained)

If current.type == ‘book’

**ToAdd.**add(**current**.collection\_papers)

If PUBLICATION\_TYPES.in(**current**.type)

**ToAdd.**add(**current**.authors)

**ToAdd.**add(**current**.editors)

**ToAdd.**add(**current**.citations)

**ToAdd.**add(**current**.works\_cited\_by)

For each **object** in **ToAdd**

If NOT (visited.isIn(**object.key** ) OR visiting.isIn(**object.key**))

visiting.add(**object.key**)

**visiting**.remove(**current**.key);

**visited**.add(**current**.key);

**Current** = **visiting**.next();

} While visiting != NULL

For each **item** in Publishing or Person data structure

If NOT **visited**.isIn(**item**.key)

return false

return true;