Materials:

In this project, we investigate the emails exchanged between employees of a company named *Barracuda National (BN)* aiming to extract knowledge out of the employee's communications with each other. The data set is a text file including 625 rows each include an employee's name E_i and names of other employees who received an email from E_i at least one time during December 2017. To process the graph we use Python programming language as well as *NetworkX* as a powerful library facilitating working with graphs.

Pre-processing:

We map the dataset from string to integer or float numbers. We extract all unique names (1000 names in total) and assign each of them a unique integer code and then replace the names with their relevant code in the original data set. The resulted dataset has the same length and dimension as the original data set except it includes integer codes instead of strings.

Overview of the network

The graph include ??? nodes and ?? edges

Figure 1 shows the graph in which red points are nodes and black parts show the edges. Since the graph contains many nodes and edges there are many overlapped areas in the figure. Figure 2 and 3 show histogram of in-degree and out-degree of the nodes. In Figure 2 (or 3) x-axis is the in-degree (or out-degree) number and y-axis is the number of nodes with relevant in-degree (or out degree).

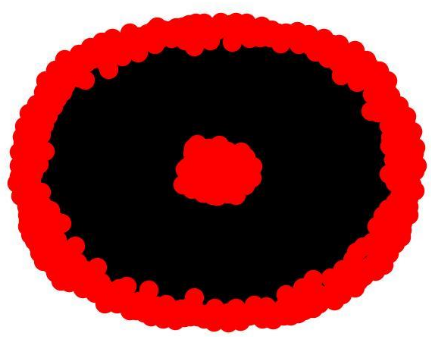

Figure 1. **The graph.** Red points are nodes which indicate employees of BN and black part is edges which shows who emailed who.

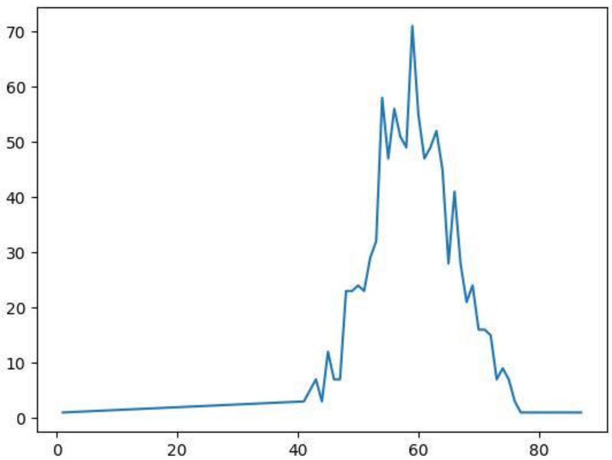

Figure 2. **Histogram of in-degree.** A point like h(60, 70) means there are 70 nodes in the graph with in-degree 60.

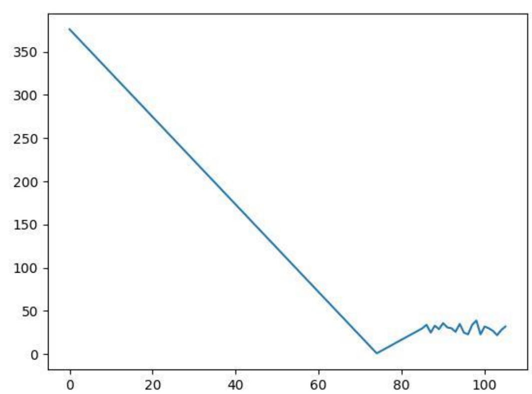

Figure 3. **Histogram of out-degree.** A point like *h (60, 50)* means there are 50 nodes in the graph with out-degree 60.

Answer to question 1: List of possible leakers

As it is describes in the question 1, a leaker is someone who probably negotiated with *Woods* which means the leaker, whoever he/she is, signaled *Wood* by email during December 2017. Therefore, we process the graph to find nodes which has an edge toward *Wood* (emailed *Wood* at least once) and also an edge back from *Wood*. In fact, if an employee E_i emailed *Wood* and *Wood* also emailed E_i at least once we consider E_i as a leaker. Our python code for implementing this idea is provided in Appendix A and the output of this code is shown in Figure 4.

```
In [2]: runfile('C:/Courses/Data Base/Project2/Graph-Processing-CS505-master/Graph-
Processing-CS505-master/leaker.py', wdir='C:/Courses/Data Base/Project2/Graph-Processing-
CS505-master/Graph-Processing-CS505-master')
Number of nodes and edges: 1000 95117
Number of leakers: 10
Leakers:
BANKS HOWE MORENO MORTON BRANCH RICHARDSON AYERS HUDSON WEST MCDANIEL
```

Figure 4. **List of leakers.** First, the number of nodes and edges are printed; then the number of leakers (10) is printed and the name of leakers is printed at the end.

Answer to question 2: List of all cliques of largest size

Finding cliques of largest size in an undirected graph is simpler to implement compared to finding them in a directed graph. We first describe how to convert the undirected graph *G* to a graph *G'* so that the converted graph satisfies two constrains: 1) *G'* is a directed graph, 2) Cliques of largest size in *G'* is equivalent to cliques of largest size in *G*. To create *G'* , we brows all pairs of nodes in *G* such as *u* and *v* and if there exist both edges *(u, v)* and *(v, u)* in graph *G* then we add an undirected edge (u, v) to *G'*. **Algorithm 1** describes how we implanted this idea.

Algorithm 1- Convert a directed graph *G* to an undirected one *G'*

```
G'. nodes = G. nodes
For all u in G. nodes
    For all v in G. nodes
        if u != v and G. has_edge(u, v) and G. has_edge(v, u)
            add (u, v) to G'. edges
```

Graph *G'* connects two nodes u and v if and only if there are two edges (u, v) and (v, u) in *G*. Since all nodes are connected to each other in a clique, if the cliques in both *G* and *G'* are the same. Therefore finding list of all cliques of largest size in *G* is equivalent to finding them in *G'*. Then, we use NetworkX library to recursively find list of all cliques of largest sizes. The whole python code is provided in Appendix A. Figure 5 shows a short part of the output of the whole program which finds the list of all maximum sized cliques.

```
In [7]: runfile('C:/Courses/Data Base/Project2/Graph-Processing-CS505-master/Graph-Processing-CS505-master/clique.py', wdir='C:/
Courses/Data Base/Project2/Graph-Processing-CS505-master/Graph-Processing-CS505-master')
Reloaded modules: k_clique, max_clique
# nodes G and g: 1000 1000
# edges G and g: 95117 4331


================ START: List of cliques of maximum size ==========
BENDER MCINTOSH WEAVER
BEASLEY LANG FRANK
BARKER GILMORE DANIELS
BENNETT ESPARZA ROACH
SAWYER VELASQUEZ MCLEAN
CARLSON VALENCIA BLACK
DAVID TYLER PETERSEN
CONLEY ROGERS SCHNEIDER
MCPHERSON NASH EWING
DURHAM LEBLANC VALENTINE
PETERSEN SERRANO TERRY
THOMPSON DUNCAN CHUNG
THOMPSON VARGAS GARRISON
```

Figure 4. **List of cliques of largest size.** In this figure first couple of cliques of largest size are shown.

The entire list of all cliques of largest size is provided below (114 cliques); each line is a clique of largest size.

BENDER MCINTOSH WEAVER

BEASLEY LANG FRANK

BARKER GILMORE DANIELS

BENNETT ESPARZA ROACH

SAWYER VELASQUEZ MCLEAN

CARLSON VALENCIA BLACK

DAVID TYLER PETERSEN

CONLEY ROGERS SCHNEIDER

MCPHERSON NASH EWING

DURHAM LEBLANC VALENTINE

PETERSEN SERRANO TERRY

THOMPSON DUNCAN CHUNG

THOMPSON VARGAS GARRISON

VELASQUEZ GARRETT ALEXANDER

HUBER OBRIEN MCCLAIN

ROBLES GILBERT CHERRY

RODRIGUEZ MATHEWS MALONE

RILEY MONTOYA SANCHEZ

RILEY DELGADO CERVANTES

SWEENEY COMBS HENDERSON

HEBERT WISE MATA

HEBERT JOSEPH AYALA

WISE TERRY MCKINNEY

MAYO FERRELL GATES

GARNER NOBLE HANNA

PALMER WEBER DAVILA

TURNER ALVAREZ REID

CROSS ORTEGA HENDRIX

CROSS PACE STEVENS

ZUNIGA DORSEY RUSSO

SANTANA GILMORE VAZQUEZ

BUCKLEY DAVILA KELLER

GOOD MCCONNELL MILLER

GOOD NOBLE HANNA

THOMAS WU CALDWELL

PONCE VAZQUEZ SCHWARTZ

HARRINGTON NORTON SAVAGE

TATE BROOKS JOHNS

TATE ATKINSON ROACH

JAMES CARNEY MATTHEWS

JAMES CHANDLER DUDLEY

SHAFFER DELACRUZ SHEPARD

DORSEY STOUT BELL

MCCARTY GARRISON VARGAS

MCCARTY GARRISON HESS

AYERS WOODS BANKS

AYERS BLACK WYATT

ZAMORA SAVAGE MUELLER

COOLEY CANNON HARMON

MUELLER STARK DANIELS

CALDWELL RITTER BUTLER

MORRIS PATRICK DUFFY

VALDEZ SALAS LAMBERT

DOUGLAS RITTER LOZANO

DOUGLAS POLLARD SIMMONS

MARTIN JACOBSON FREEMAN

JOHNSTON HUNTER MCCOY

JOHNSTON HUBBARD MALDONADO

FOX MELTON HENDERSON

WALTON MADDOX ROY

WALTON MADDOX BROWN

LONG WOOD SANFORD

PRICE LANG HERNANDEZ

HULL SIMON GALLOWAY

JENSEN HUNT PERRY

JENSEN HUNT NICHOLSON

DILLON COWAN COOKE

BAIRD COCHRAN LOVE

YOUNG BECKER DOMINGUEZ

PETERSON VELAZQUEZ FLEMING

PETERSON CUEVAS MCBRIDE

KEMP MERCADO BELL

MCCLAIN RAYMOND WELLS

RIVERA FLETCHER STEELE

PATRICK DUFFY MCKAY

MERCER MCGRATH SOLOMON

MERCER MCGRATH BLAKE

GRAHAM OCHOA DUKE

MEJIA MCKEE GILLESPIE

MILES GARRETT ALEXANDER

BRAUN TANNER LIN

WEAVER ARELLANO AGUIRRE

DOUGHERTY MORENO HINES

ANDRADE MATHEWS WALSH

KOCH SHIELDS JACOBSON

BYRD BRENNAN ROBERTS

HUYNH CORTEZ COMPTON

BANKS WOODS MORENO

CARR LOZANO GOLDEN

RITTER LOZANO BRANCH

SANTOS STARK BURTON

SANTOS ERICKSON MCCALL

WADE BATES MELENDEZ

DODSON GREER MCCALL

CHERRY BARNETT ELLIS

CHERRY BARNETT SANCHEZ

KENT BOND HOUSE

MATHEWS GUTIERREZ WU

ROBINSON WYATT COBB

POTTS PENA POWELL

LOZANO WILKINSON WONG

LOWE FARMER CONTRERAS

WEISS WOOD SILVA

ELLIOTT WYATT MCGEE

ARELLANO CONWAY POWERS

MELTON WYATT STEVENS

STEPHENS RAMIREZ PUGH

ANDREWS GREER VELEZ

FARLEY LIU ZAVALA

ZHANG GOODWIN SAVAGE

FARMER REILLY KELLEY

LAM EDWARDS COBB

VALENZUELA GUERRA PROCTOR

BISHOP VILLARREAL SAUNDERS

Answer to question 3: Find a butterfly in the graph

To answer this question we benefit from the list of cliques we already found while solving question 2. We brows all possible pairs of cliques such as C_i and C_j and then if they both have length 3 and are connected in one point we consider them as a butterfly. **Algorithm 2** describes how we implanted this idea. The python code to implement Algorithm 2 is included in Appendix A.

Algorithm 2- Finding butterflies

list of cliques= Algorithm 1(G)
For all cliques C_i in list of cliques
    For all cliques C_j in list of cliques
        Butterfly_Candidate = C_i . nodes + (C_j *.nodes* – C_i *.nodes*)
        if C_j *!=* C_j and *length(Butterfly_Candidate) == 5*
            butterfly_list.append(Butterfly_Candidate)

Answer Q3:

Answer Q4: