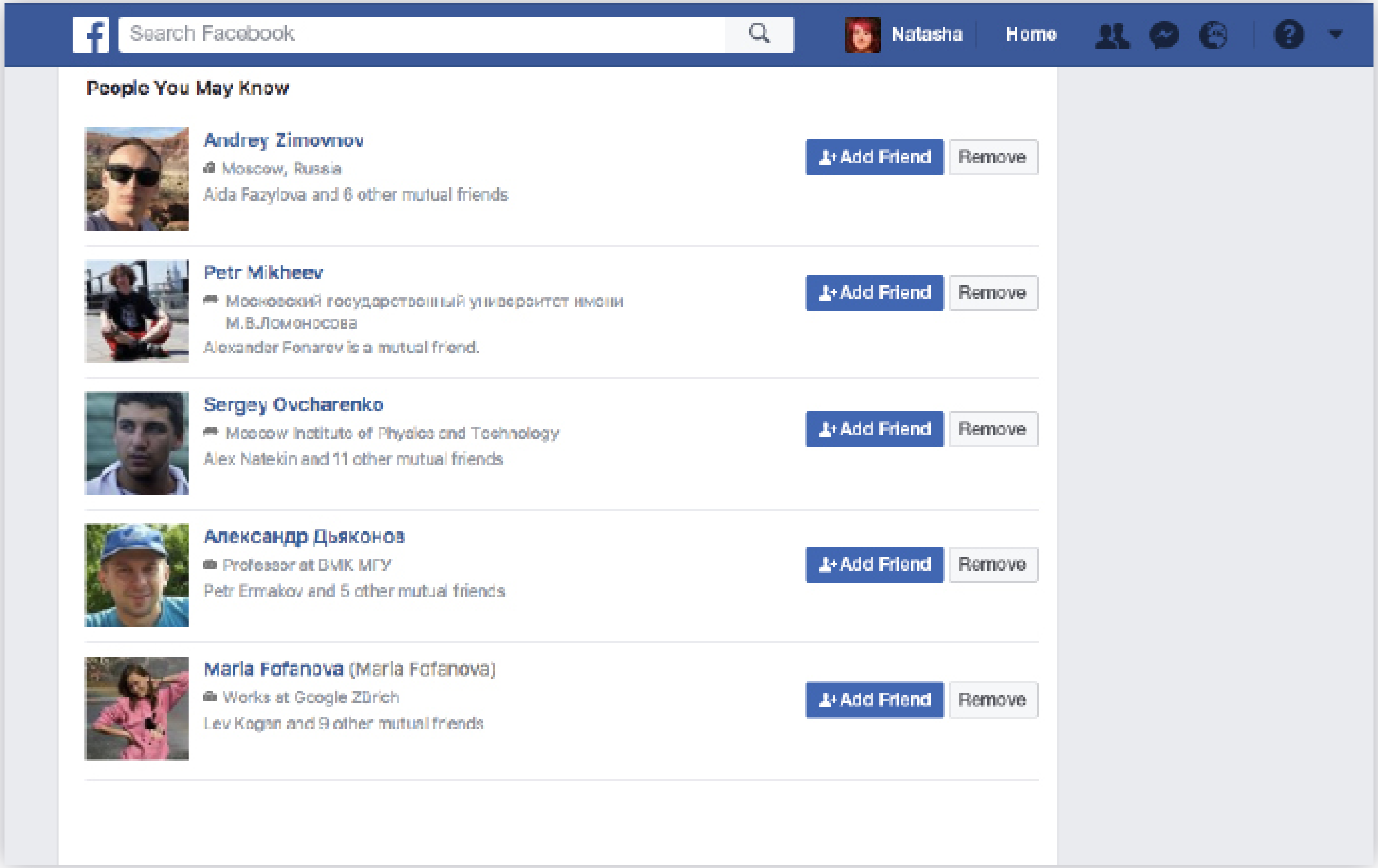
Counting common friends



Search Facebook

Natasha Home

People You May Know

Andrey Zimovnov
Moscow, Russia
Aida Fazylova and 6 other mutual friends

Add Friend Remove

Petr Mikheev
Московский государственный университет имени М.В.Ломоносова
Alexander Fonarev is a mutual friend.

Add Friend Remove

Sergey Ovcharenko
Moscow Institute of Physics and Technology
Alex Natekin and 11 other mutual friends

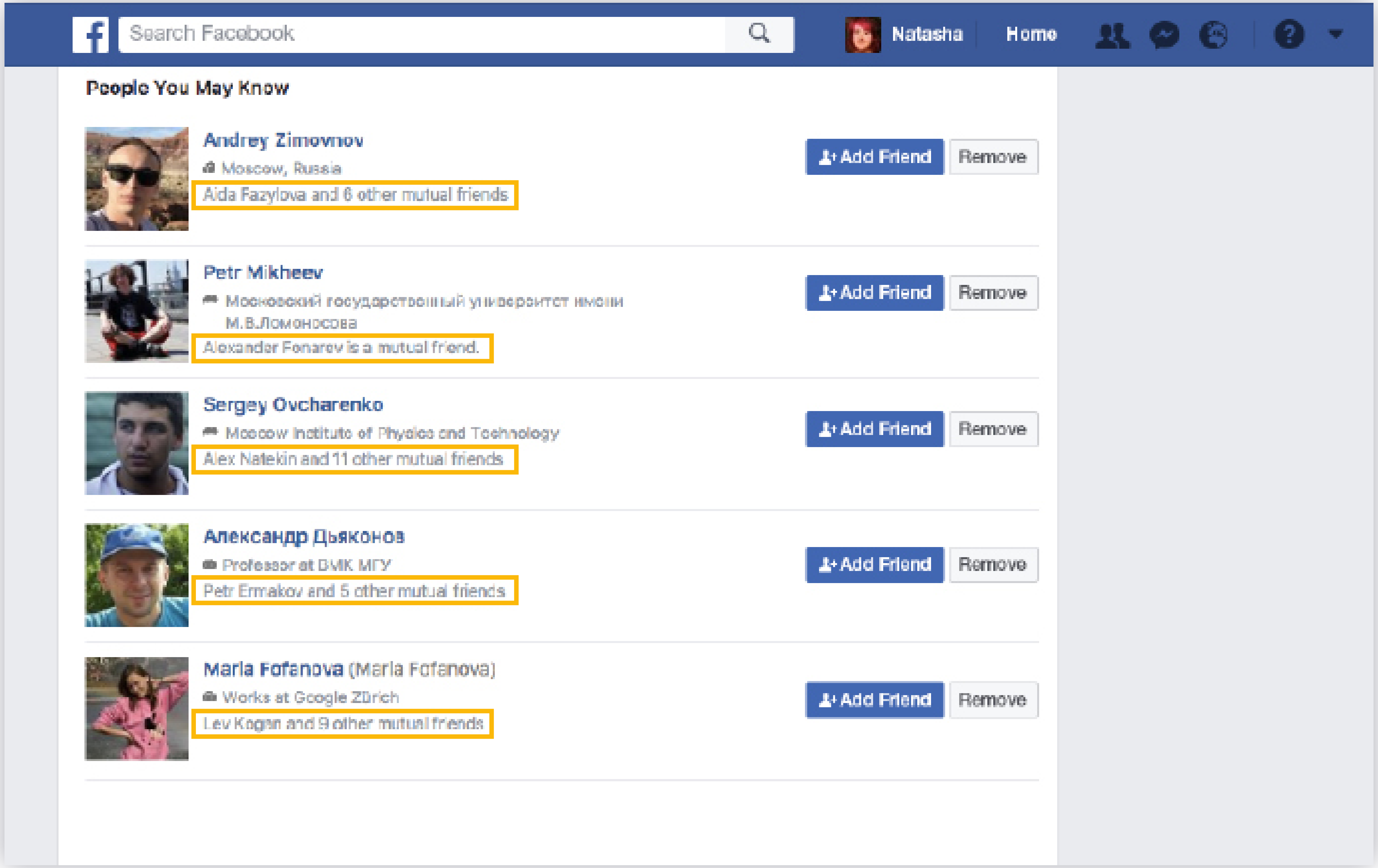
Add Friend Remove

Александр Дьяконов
Professor at ВМК МГУ
Petr Ermakov and 5 other mutual friends

Add Friend Remove

Maria Fofanova (Maria Fofanova)
Works at Google Zürich
Lev Kogan and 9 other mutual friends

Add Friend Remove



People You May Know

Andrey Zimovnov
Moscow, Russia
Aida Fazylova and 6 other mutual friends

Petr Mikheev
Московский государственный университет имени М.В.Ломоносова
Alexander Fonarev is a mutual friend.

Sergey Ovcharenko
Moscow Institute of Physics and Technology
Alex Natekin and 11 other mutual friends

Александр Дьяконов
Professor at ВМК МГУ
Petr Ermakov and 5 other mutual friends

Maria Fofanova (Maria Fofanova)
Works at Google Zürich
Lev Kogan and 9 other mutual friends

SNA hackathon 2016

40 000 000 day active users

Adjacency matrix?

Adjacency matrix?

40 000 000 x 40 000 000

```
0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0
0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0
0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0
0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0
0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0
1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1
0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0
0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0
```

Adjacency matrix?

40 000 000 x 40 000 000

```
0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0
0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0
0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0
0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0
0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0
1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1
0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0
0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0
```

40 000 000 x 500 x 2 ones in adjacency matrix

Adjacency matrix?

40 000 000 x 40 000 000

```
0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0
0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0
0 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0
0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0
0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0
1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 1
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1
0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0
0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0
```

40 000 000 x 500 x 2 ones in adjacency matrix

only 1 / 40 000

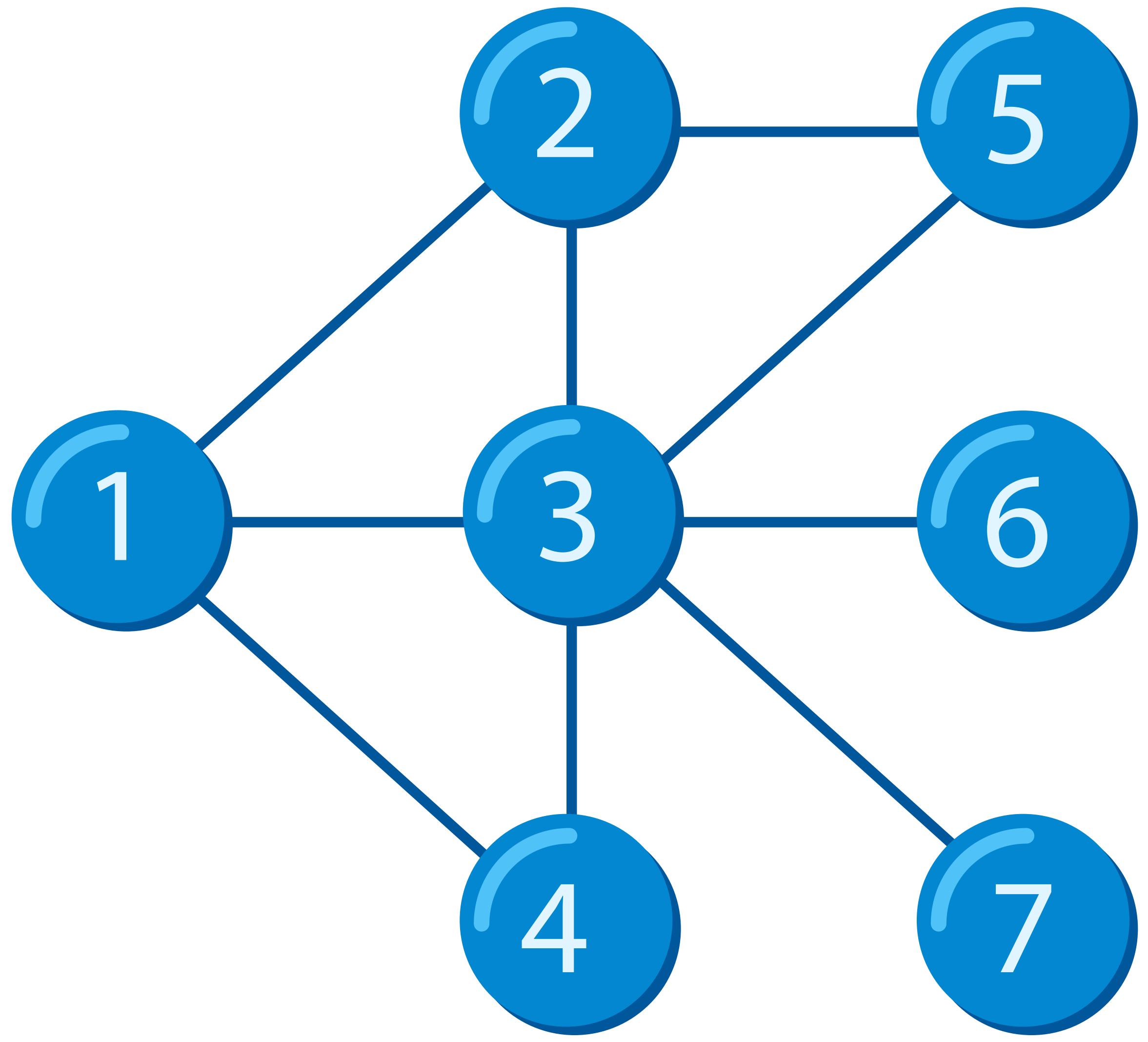
Edge list?

Adjacency lists?

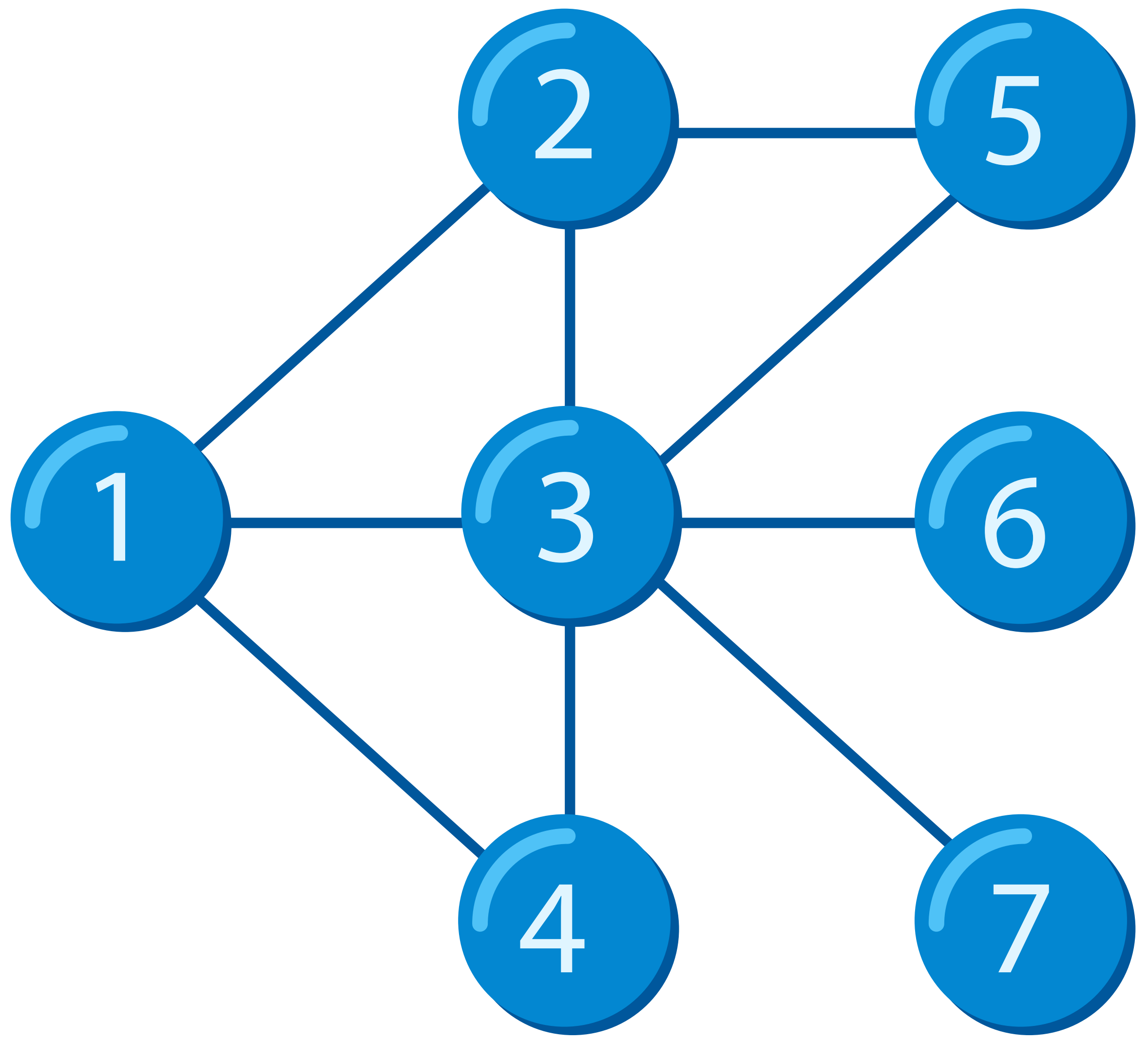
$\boxed{\text{Edge list?}}$

Adjacency lists?

Mini social graph



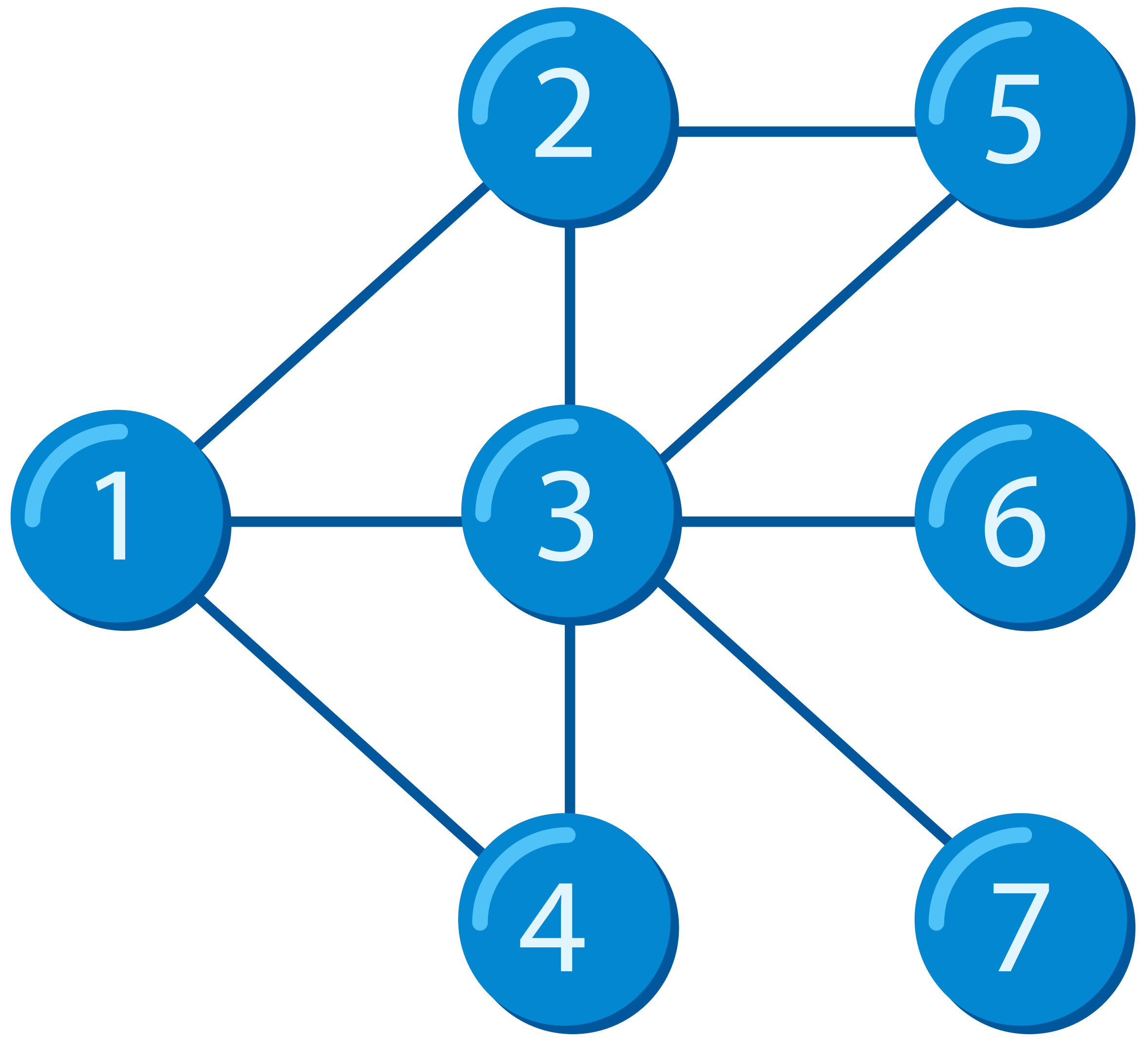
Mini social graph



Edge list

<1, 2>
<1, 3>
<1, 4>
<2, 3>
<2, 5>
<3, 4>
<3, 5>
<3, 6>
<3, 7>

Mini social graph

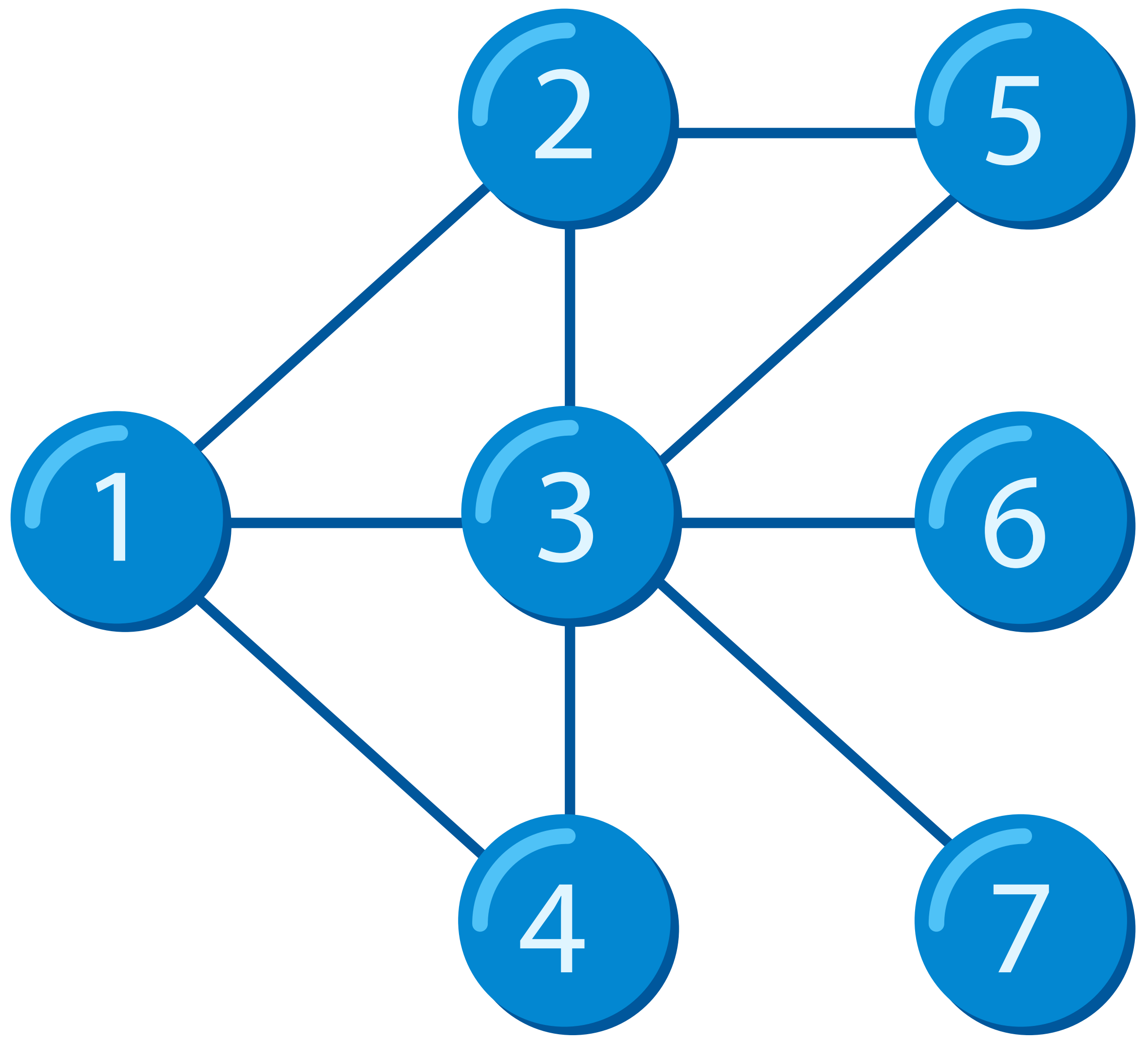


Edge list

Dataframe

<1, 2>
<1, 3>
<1, 4>
<2, 3>
<2, 5>
<3, 4>
<3, 5>
<3, 6>
<3, 7>

Mini social graph

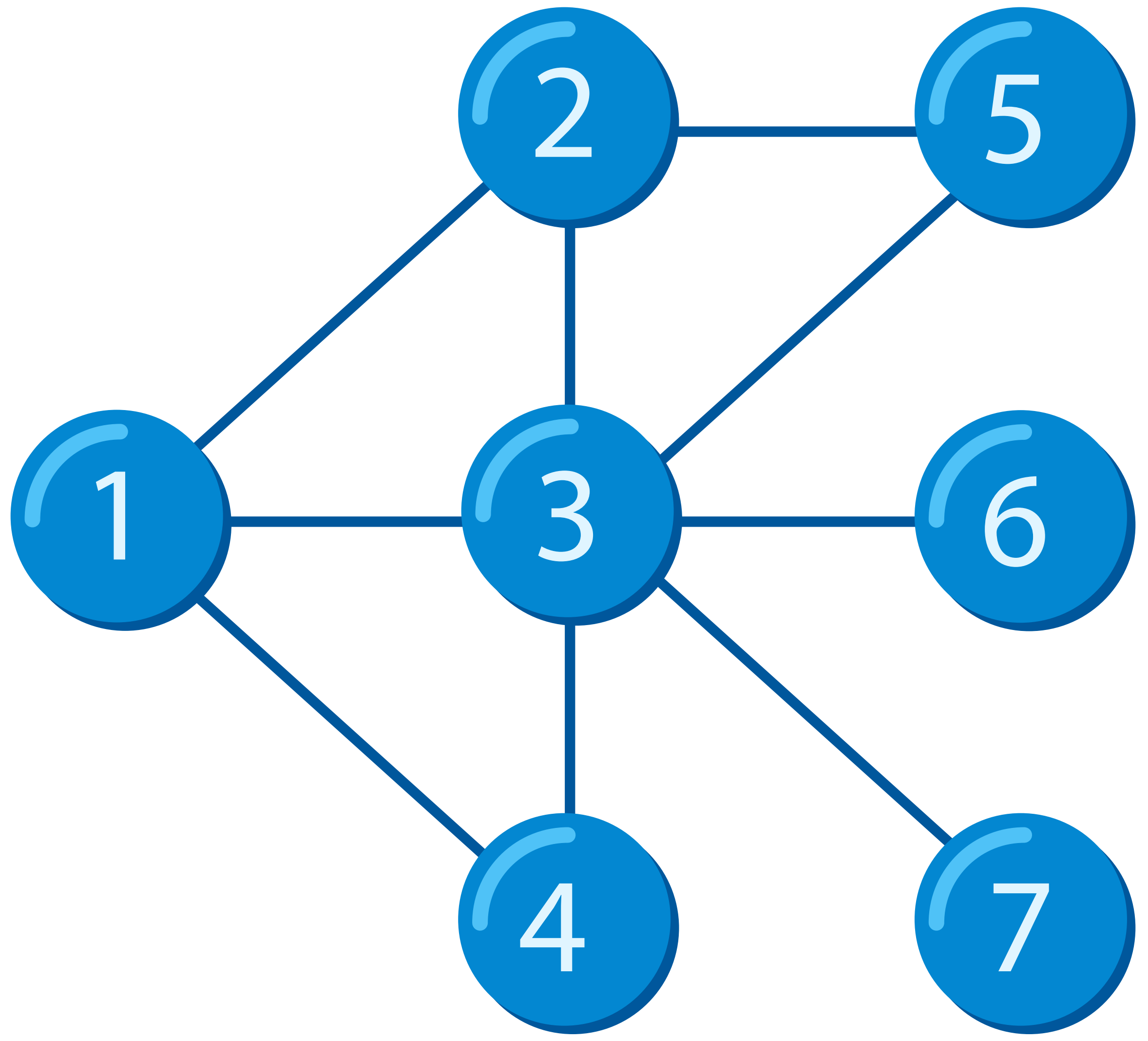


Edge list

Dataframe A

<1, 2>
<1, 3>
<1, 4>
<2, 3>
<2, 5>
<3, 4>
<3, 5>
<3, 6>
<3, 7>

Mini social graph



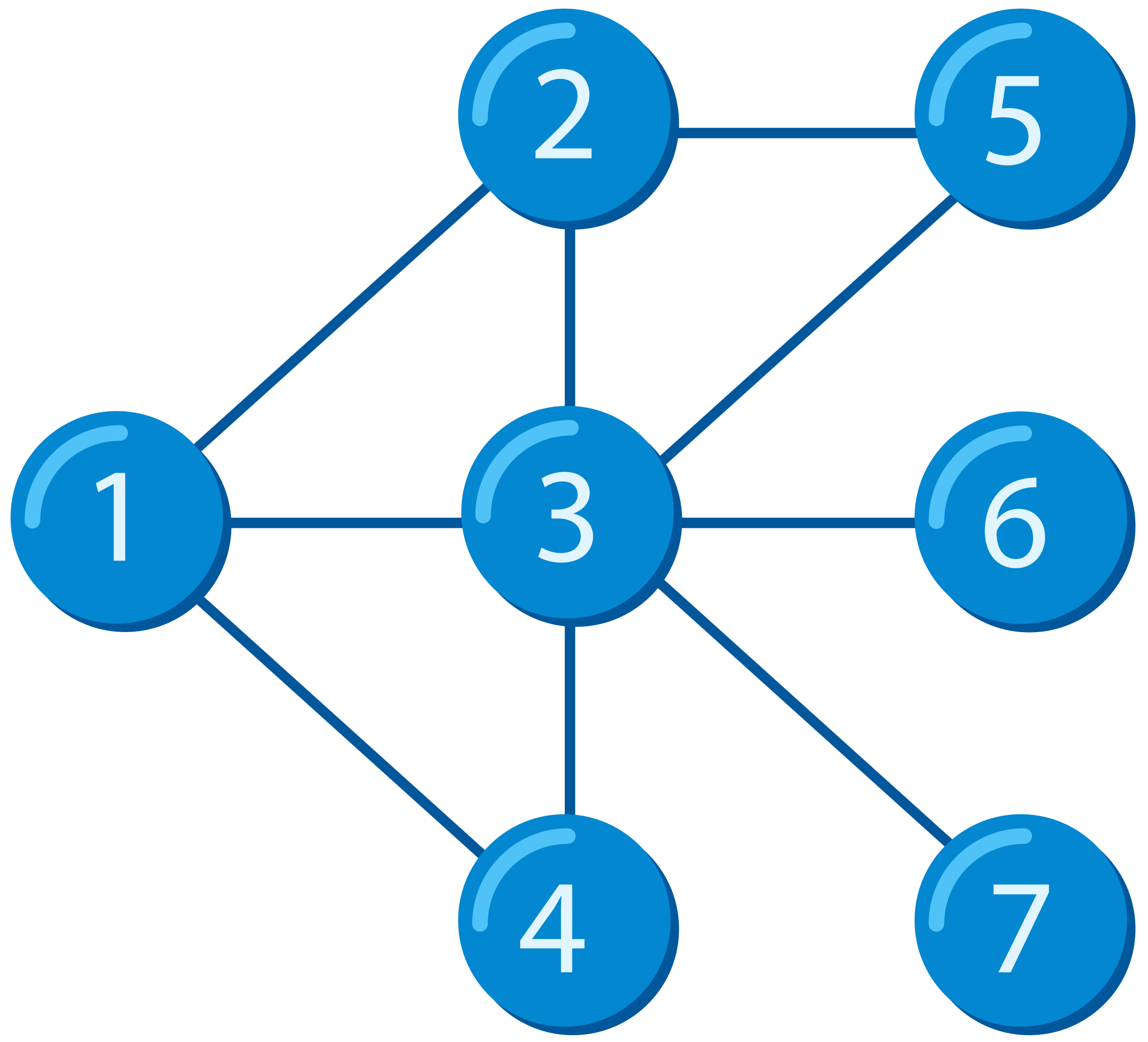
Edge list

Dataframe

A B

<1, 2>
<1, 3>
<1, 4>
<2, 3>
<2, 5>
<3, 4>
<3, 5>
<3, 6>
<3, 7>

Mini social graph



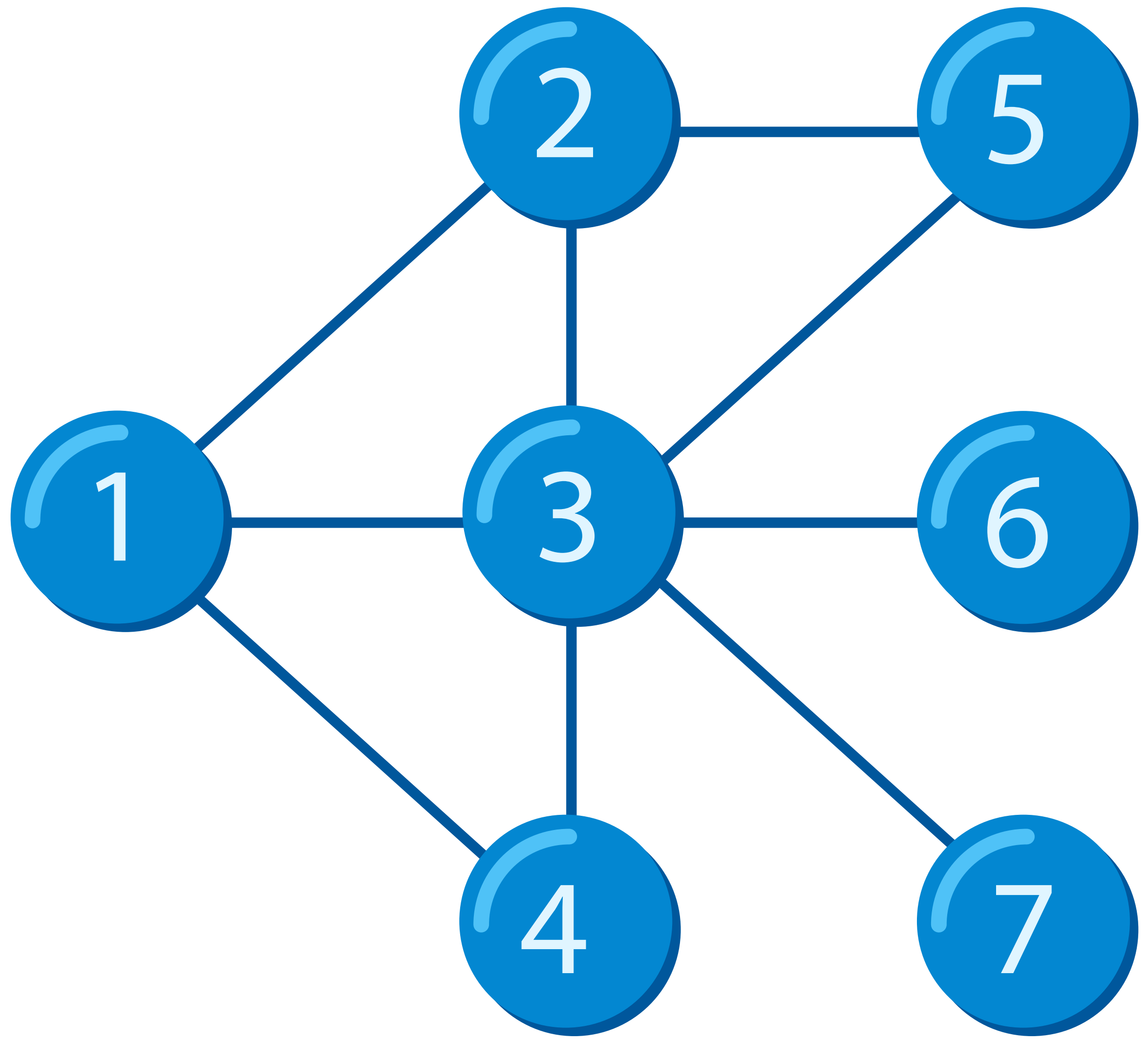
Edge list

Dataframe

A B

<1, 2>
<1, 3>
<1, 4>
<2, 3>
<2, 5>
<3, 4>
<3, 5>
<3, 6>
<3, 7>

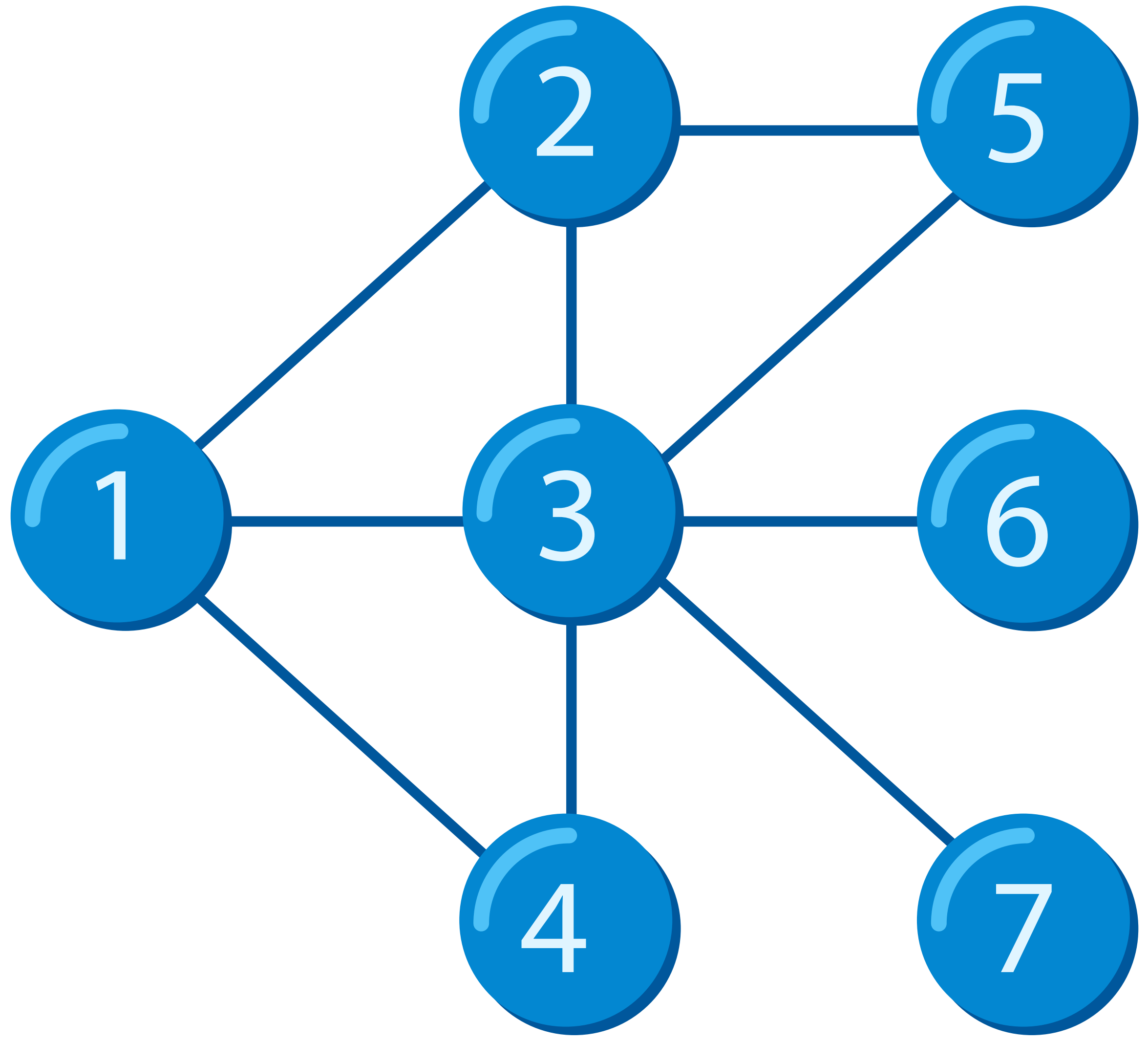
Mini social graph



Common friends for user 1

<(1, 2), 1> - user 3
<(1, 3), 2> - users 2 & 4
<(1, 4), 1> - user 3
<(1, 5), 2> - users 2 & 3
<(1, 6), 1> - user 3
<(1, 7), 1> - user 3

Combinatorial explosion

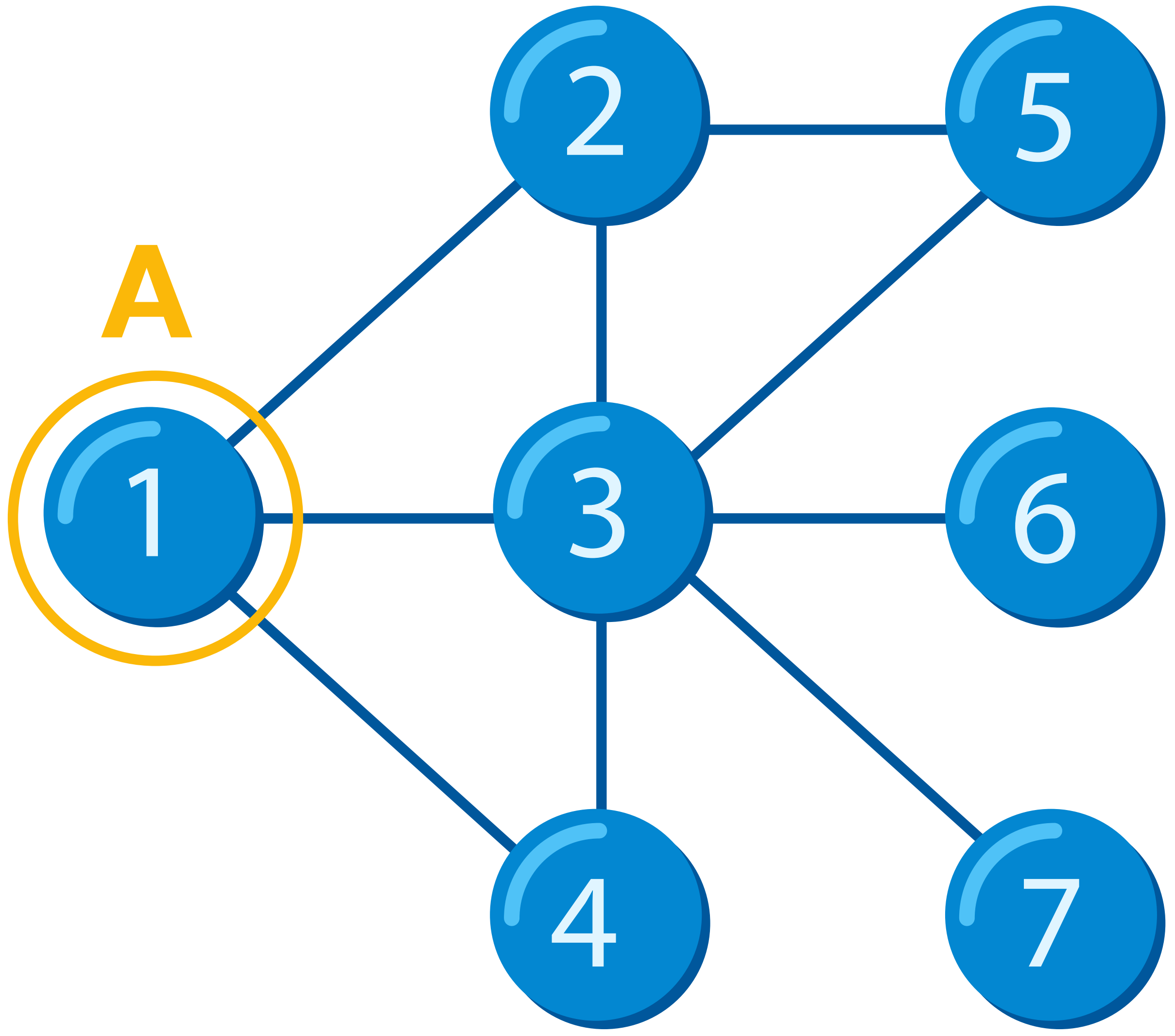


<1, 2>
<1, 3>
<1, 4>

VS

<(1, 2), 1>
<(1, 3), 2>
<(1, 4), 1>
<(1, 5), 2>
<(1, 6), 1>
<(1, 7), 1>

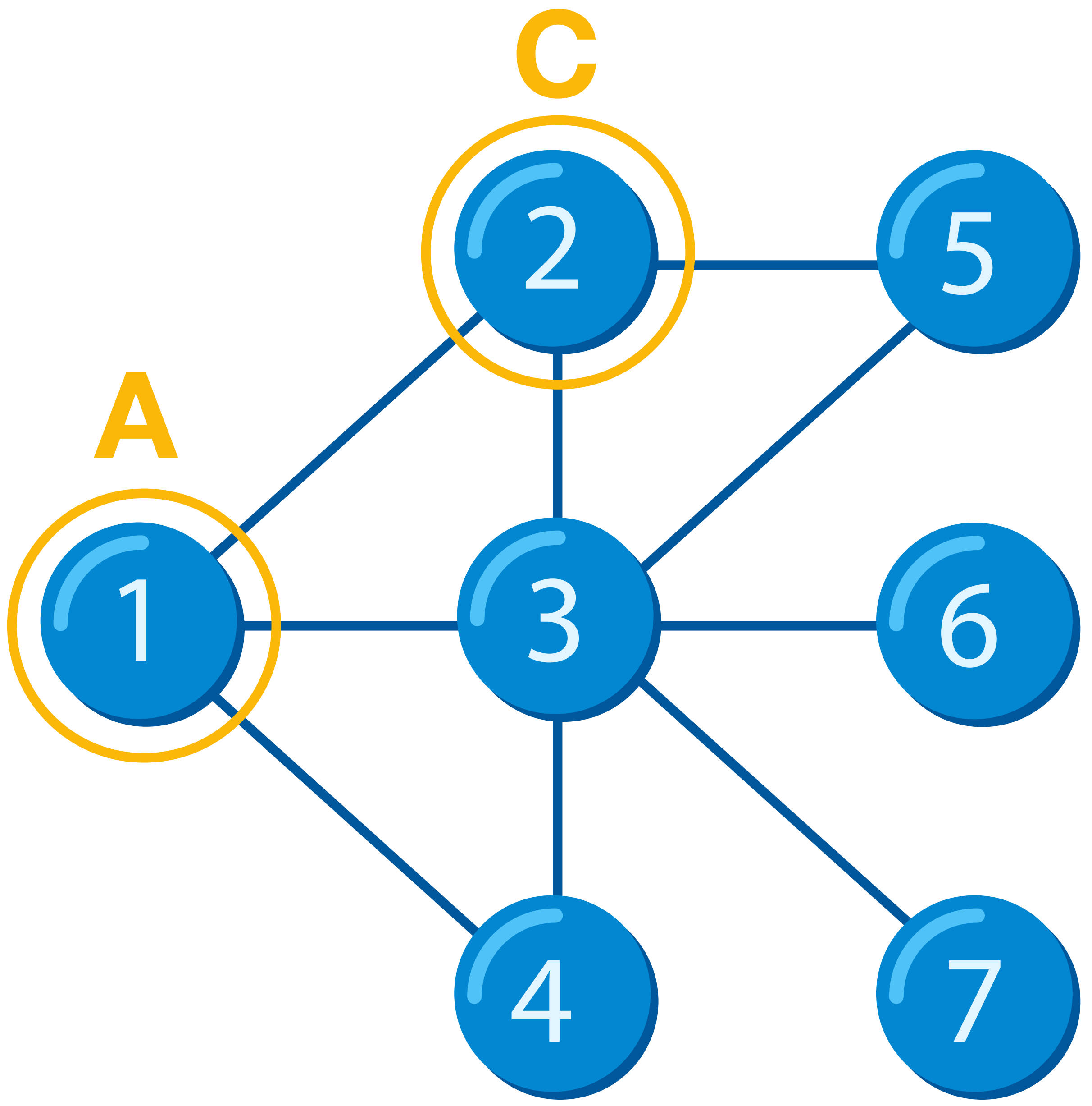
Mini social graph



Common friends for user 1

<(1, 2), 1> - user 3
<(1, 3), 2> - users 2 & 4
<(1, 4), 1> - user 3
<(1, 5), 2> - users 2 & 3
<(1, 6), 1> - user 3
<(1, 7), 1> - user 3

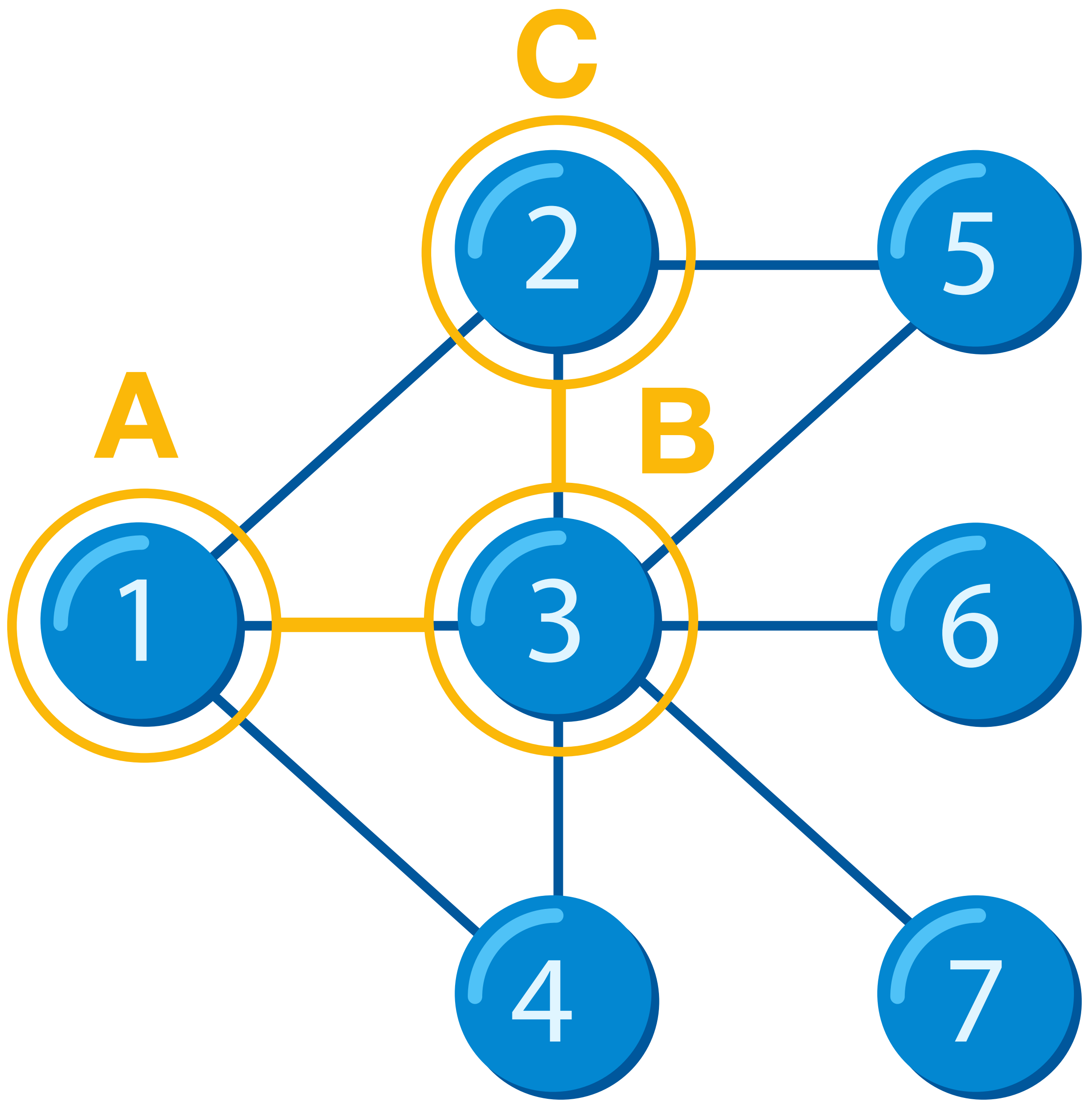
Mini social graph



Common friends for user 1

<(1, 2), 1> - user 3
<(1, 3), 2> - users 2 & 4
<(1, 4), 1> - user 3
<(1, 5), 2> - users 2 & 3
<(1, 6), 1> - user 3
<(1, 7), 1> - user 3

Mini social graph

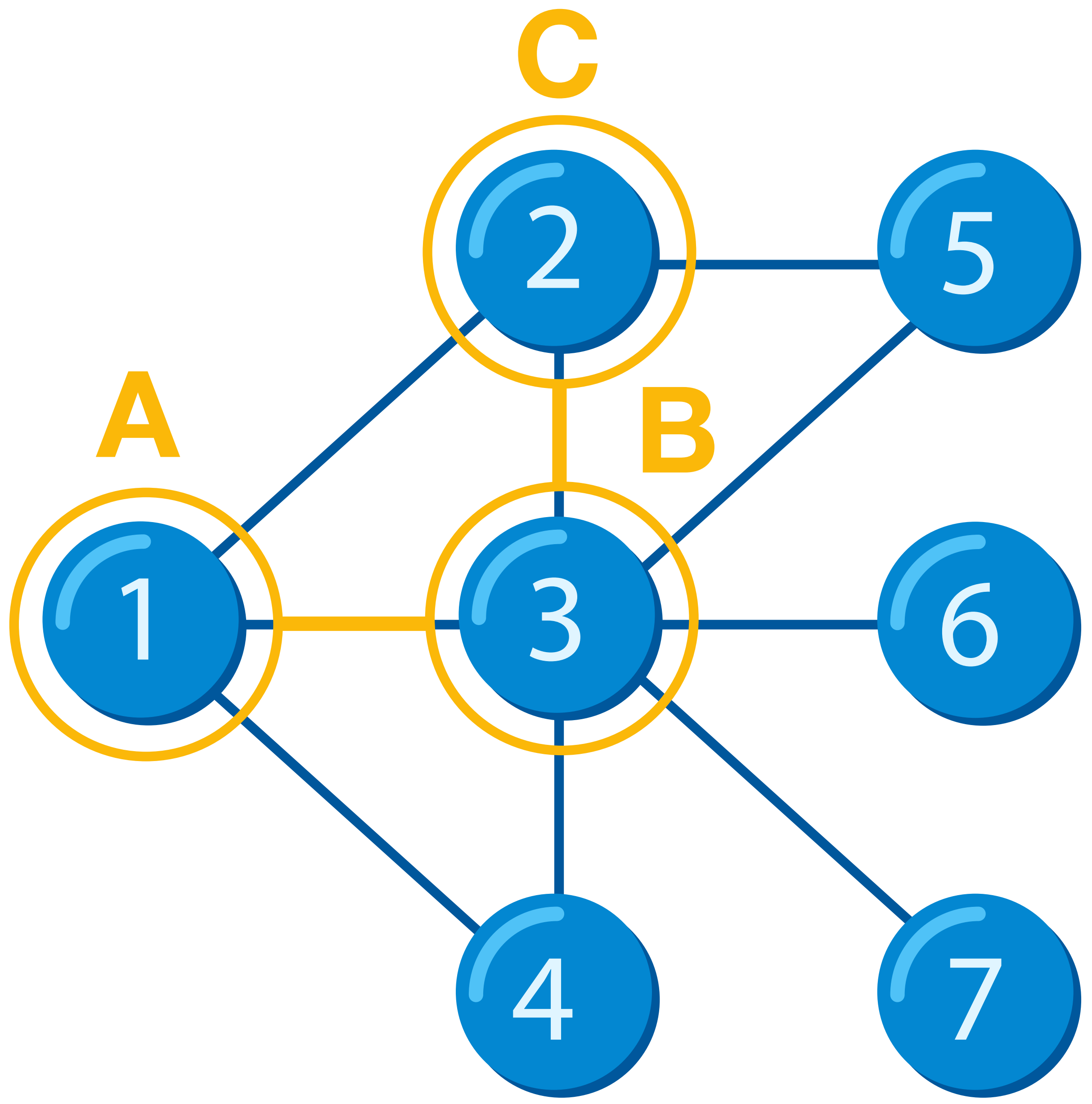


Common friends for user 1

<(1, 2), 1> - user 3
<(1, 3), 2> - users 2 & 4
<(1, 4), 1> - user 3
<(1, 5), 2> - users 2 & 3
<(1, 6), 1> - user 3
<(1, 7), 1> - user 3

Dataframe	
A	B
<1, 2>	
<1, 3>	
<1, 4>	
<2, 3>	
<2, 5>	
<3, 4>	
<3, 5>	
<3, 6>	
<3, 7>	

Dataframe	
A	B
<1, 2>	
<1, 3>	
<1, 4>	
<2, 3>	
<2, 5>	
<3, 4>	
<3, 5>	
<3, 6>	
<3, 7>	

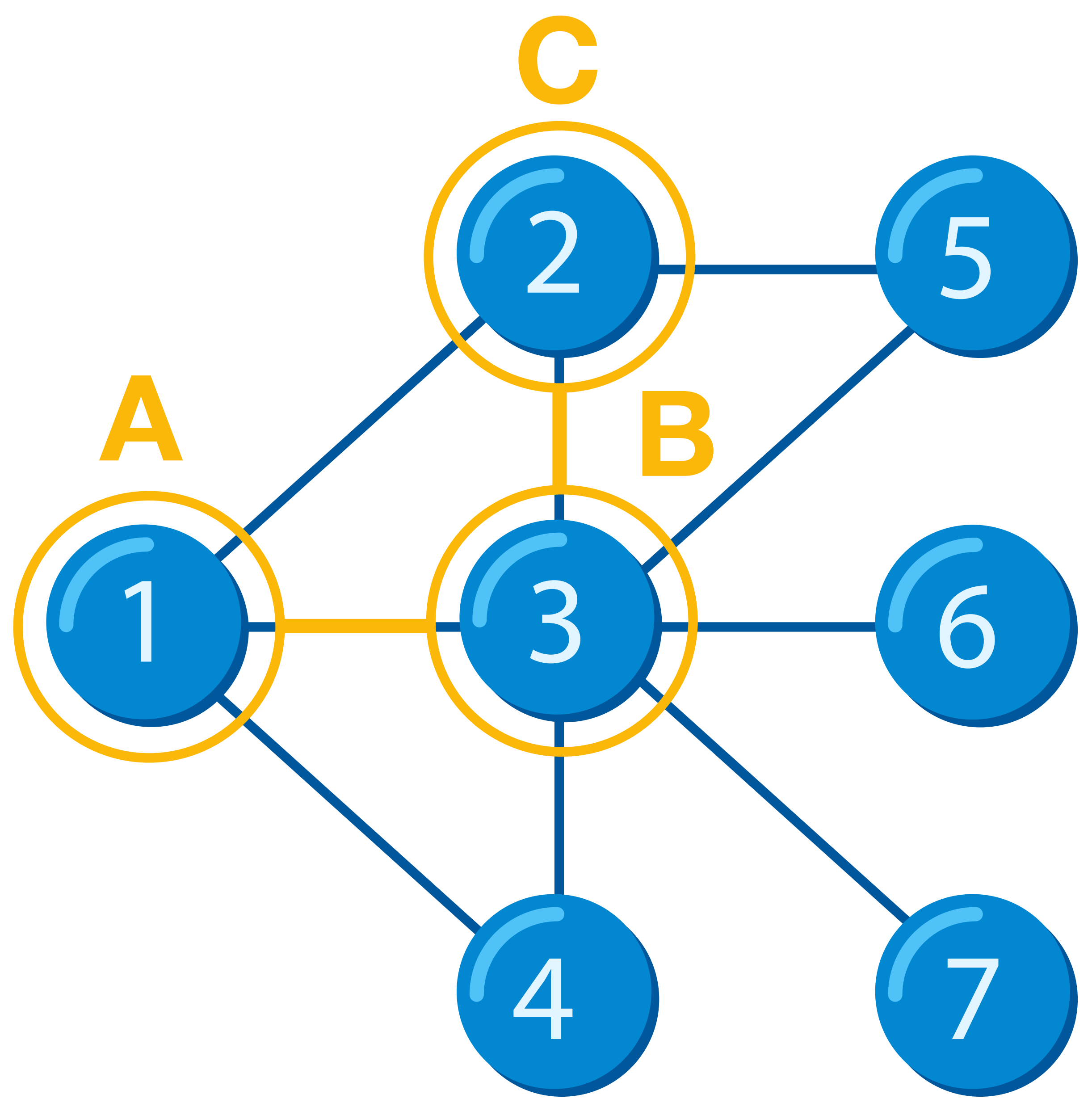


Dataframe

A	B
<1,	2>
<1,	3>
<1,	4>
<2,	3>
<2,	5>
<3,	4>
<3,	5>
<3,	6>
<3,	7>

Dataframe

B	C
<1,	2>
<1,	3>
<1,	4>
<2,	3>
<2,	5>
<3,	4>
<3,	5>
<3,	6>
<3,	7>



Dataframe

A B

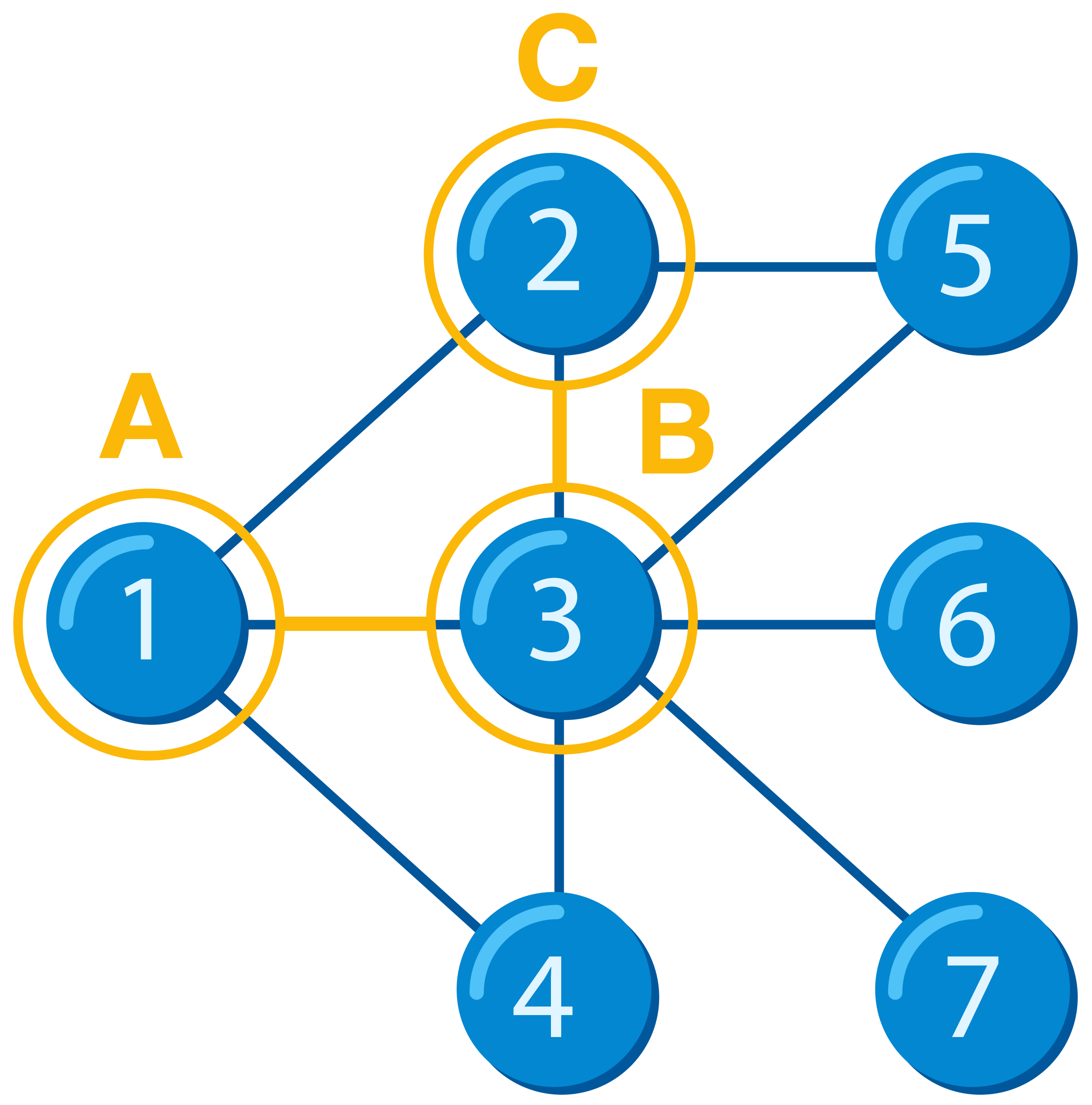
<1, 2>
<1, 3>
<1, 4>
<2, 3>
<2, 5>
<3, 4>
<3, 5>
<3, 6>
<3, 7>

join on

Dataframe

B C

<1, 2>
<1, 3>
<1, 4>
<2, 3>
<2, 5>
<3, 4>
<3, 5>
<3, 6>
<3, 7>



```python
from pyspark.sql import SparkSession
sparkSession = SparkSession.builder.enableHiveSupport().master("local").getOrCreate()

from pyspark.sql.types import StructType, StructField, IntegerType, StringType
from pyspark.sql import Row

EdgeList = [(1, 2), (1, 3), (1, 4), (2, 3), (2, 5), (3, 4), (3, 5), (3, 6), (3, 7)]
graphData = sparkSession.sparkContext.parallelize(edgeList).map(lambda (arc, dst):
Row(src, dst)

graphSchemaAB = StructType([StructFie1d('A', IntegerType(), nullable=False),
StructField('B', StringType(), nullable=False)])

abDF = sparkSession.createDataFrame(graphData, graphSchemaAB)
graphSchemaBCl = StructType([StructField 'B', IntegerType(), nullable=False),
StructField('C', StringType(), nullable=False)])

bcDF = sparkSession.createDataFrame(graphData, graphSchemaBCl)

abDF.show()
```

```python
from pyspark.sql import SparkSession
sparkSession = SparkSession.builder.enableHiveSupport().master("local").getOrCreate()

from pyspark.sql.types import StructType, StructField, IntegerType, StringType
from pyspark.sql import Row

EdgeList = [(1, 2), (1, 3), (1, 4), (2, 3), (2, 5), (3, 4), (3, 5), (3, 6), (3, 7)]
graphData = sparkSession.sparkContext.parallelize(edgeList).map(lambda (arc, dst):
Row(src, dst)

graphSchemaAB = StructType([StructFie1d('A', IntegerType(), nullable=False),
StructField('B', StringType(), nullable=False)])

abDF = sparkSession.createDataFrame(graphData, graphSchemaAB)
graphSchemaBCl = StructType([StructField 'B', IntegerType(), nullable=False),
StructField('C', StringType(), nullable=False)])

bcDF = sparkSession.createDataFrame(graphData, graphSchemaBCl)

abDF.show()
```

```python
from pyspark.sql import SparkSession
sparkSession = SparkSession.builder.enableHiveSupport().master("local").getOrCreate()

from pyspark.sql.types import StructType, StructField, IntegerType, StringType
from pyspark.sql import Row

EdgeList = [(1, 2), (1, 3), (1, 4), (2, 3), (2, 5), (3, 4), (3, 5), (3, 6), (3, 7)]
graphData = sparkSession.sparkContext.parallelize(edgeList).map(lambda (arc, dst):
Row(src, dst)

graphSchemaAB = StructType([StructFie1d('A', IntegerType(), nullable=False),
StructField('B', StringType(), nullable=False)])

abDF = sparkSession.createDataFrame(graphData, graphSchemaAB)
graphSchemaBCl = StructType([StructField 'B', IntegerType(), nullable=False),
StructField('C', StringType(), nullable=False)])

bcDF = sparkSession.createDataFrame(graphData, graphSchemaBCl)

abDF.show()
```

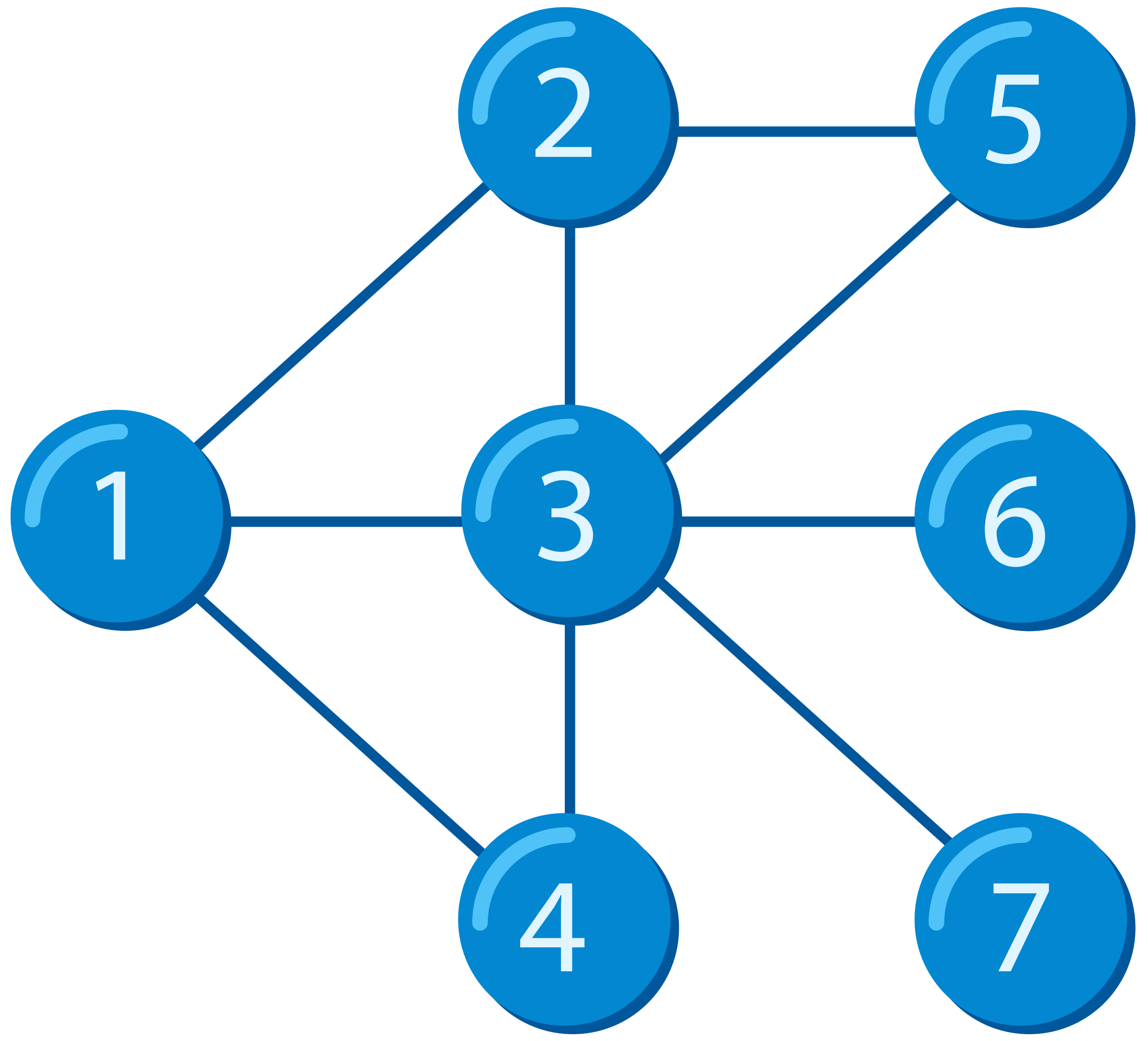
```
from pyspark.sql import SparkSession
sparkSession = SparkSession.builder.enableHiveSupport().master("local").getOrCreate()

from pyspark.sql.types import StructType, StructField, IntegerType, StringType
from pyspark.sql import Row

EdgeList = [(1, 2), (1, 3), (1, 4), (2, 3), (2, 5), (3, 4), (3, 5), (3, 6), (3, 7)]
graphData = sparkSession.sparkContext.parallelize(edgeList).map(lambda (arc, dst):
Row(src, dst)

graphSchemaAB = StructType([StructFie1d('A', IntegerType(), nullable=False),
StructField('B', StringType(), nullable=False)])

abDF = sparkSession.createDataFrame(graphData, graphSchemaAB)
graphSchemaBCl = StructType([StructField 'B', IntegerType(), nullable=False),
StructField('C', StringType(), nullable=False)])

bcDF = sparkSession.createDataFrame(graphData, graphSchemaBCl)

abDF.show()
```

```python
from pyspark.sql import SparkSession
sparkSession = SparkSession.builder.enableHiveSupport().master("local").getOrCreate()

from pyspark.sql.types import StructType, StructField, IntegerType, StringType
from pyspark.sql import Row

EdgeList = [(1, 2), (1, 3), (1, 4), (2, 3), (2, 5), (3, 4), (3, 5), (3, 6), (3, 7)]
graphData = sparkSession.sparkContext.parallelize(edgeList).map(lambda (arc, dst):
Row(src, dst)

graphSchemaAB = StructType([StructFie1d('A', IntegerType(), nullable=False),
StructField('B', StringType(), nullable=False)])

abDF = sparkSession.createDataFrame(graphData, graphSchemaAB)
graphSchemaBCl = StructType([StructField 'B', IntegerType(), nullable=False),
StructField('C', StringType(), nullable=False)])

bcDF = sparkSession.createDataFrame(graphData, graphSchemaBCl)

abDF.show()
abDF.show()
+---+---+
|  A|  B|
+---+---+
|  1|  2|
|  1|  3|
|  1|  4|
|  2|  3|
|  2|  5|
|  3|  4|
|  3|  5|
|  3|  6|
|  3|  7|
+---+---+
```

```
bcDF.show()
+---+---+
|  B|  C|
+---+---+
|  1|  2|
|  1|  3|
|  1|  4|
|  2|  3|
|  2|  5|
|  3|  4|
|  3|  5|
|  3|  6|
|  3|  7|
+---+---+
joinDF = abDF.join(bcDF, abDF.B == bcDF.B)
joinDF.show()
```

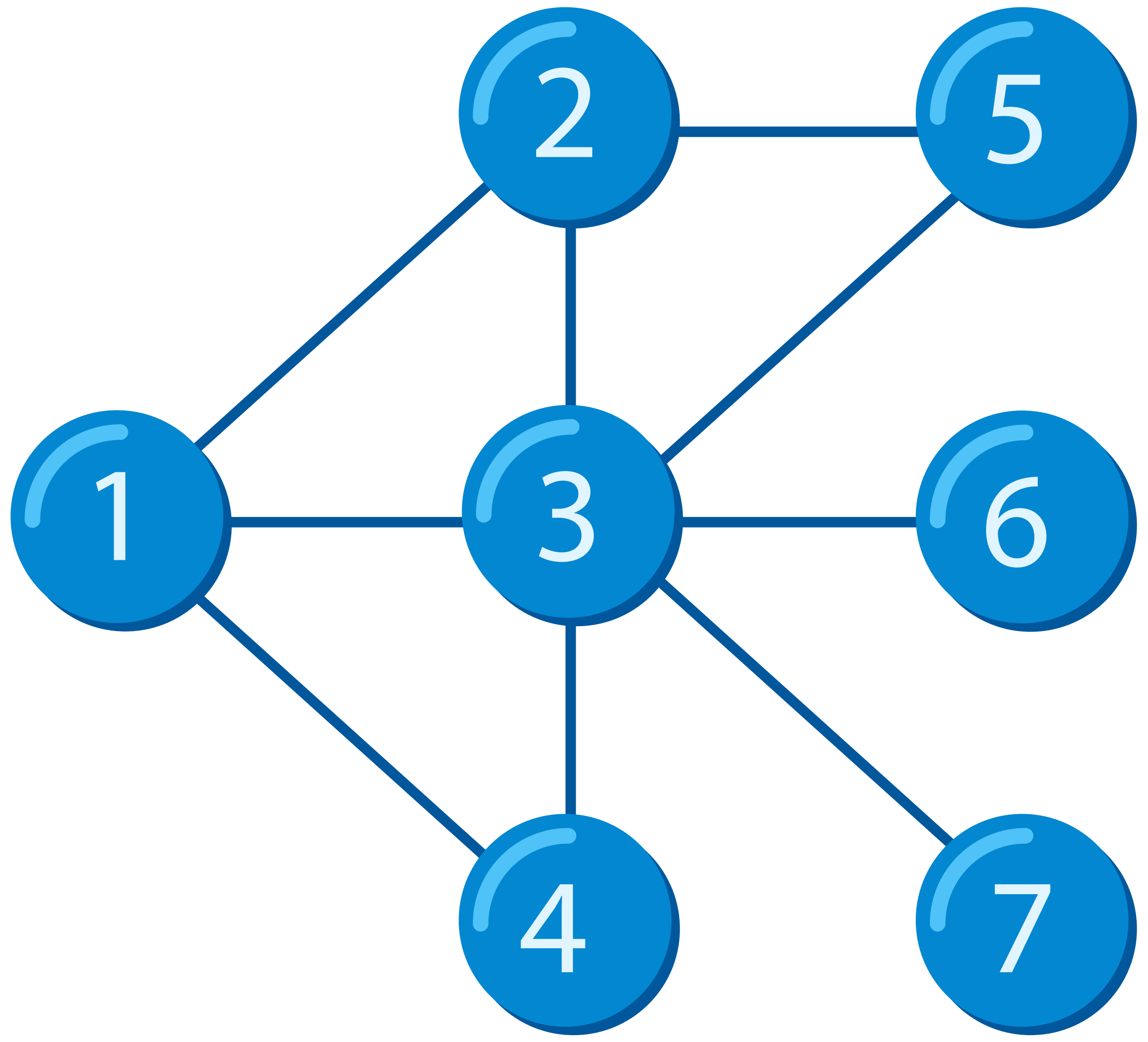
```
+---+---+---+---+
|  A|  B|  B|  C|
+---+---+---+---+
|  1|  3|  3|  4|
|  1|  3|  3|  5|
|  1|  3|  3|  6|
|  1|  3|  3|  7|
|  2|  3|  3|  4|
|  2|  3|  3|  5|
|  2|  3|  3|  7|
|  1|  2|  2|  3|
|  1|  2|  2|  5|
+---+---+---+---+
abcDF.drop("B")
    .groupBy("A", "C")
    .count()
    .filter("A = 1").show()
```

```
+---+---+-----+
|  A|  C|count|
+---+---+-----+
|  1|  3|    1|
|  1|  4|    1|
|  1|  5|    2|
|  1|  6|    1|
|  1|  7|    1|
+---+---+-----+
```



Result

```
+---+---+-----+
|  A|  C|count|
+---+---+-----+
|  1|  3|    1|
|  1|  4|    1|
|  1|  5|    2|
|  1|  6|    1|
|  1|  7|    1|
+---+---+-----+
```



Result

```
+---+---+-------+
|  A|  C|count|
+---+---+-------+
|  1|  3|    1|
|  1|  4|    1|
|  1|  5|    2|
|  1|  6|    1|
|  1|  7|    1|
+---+---+-------+
abDF.show()         bcDF.show()
+---+---+           +---+---+
|  A|  B|           |  B|  C|
+---+---+           +---+---+
|  1|  2|           |  1|  2|
|  1|  3|           |  1|  3|
|  1|  4|           |  1|  4|
|  2|  3|           |  2|  3|
|  2|  5|           |  2|  5|
|  3|  4|           |  3|  4|
|  3|  5|           |  3|  5|
|  3|  6|           |  3|  6|
|  3|  7|           |  3|  7|
+---+---+           +---+---+
   3   2               3   2
```

```
abDF.show()
+---+---+
|  A|  B|
+---+---+
|  1|  2|
|  1|  3|
|  1|  4|
|  2|  3|
|  2|  5|
|  3|  4|
|  3|  5|
|  3|  6|
|  3|  7|
+---+---+
```

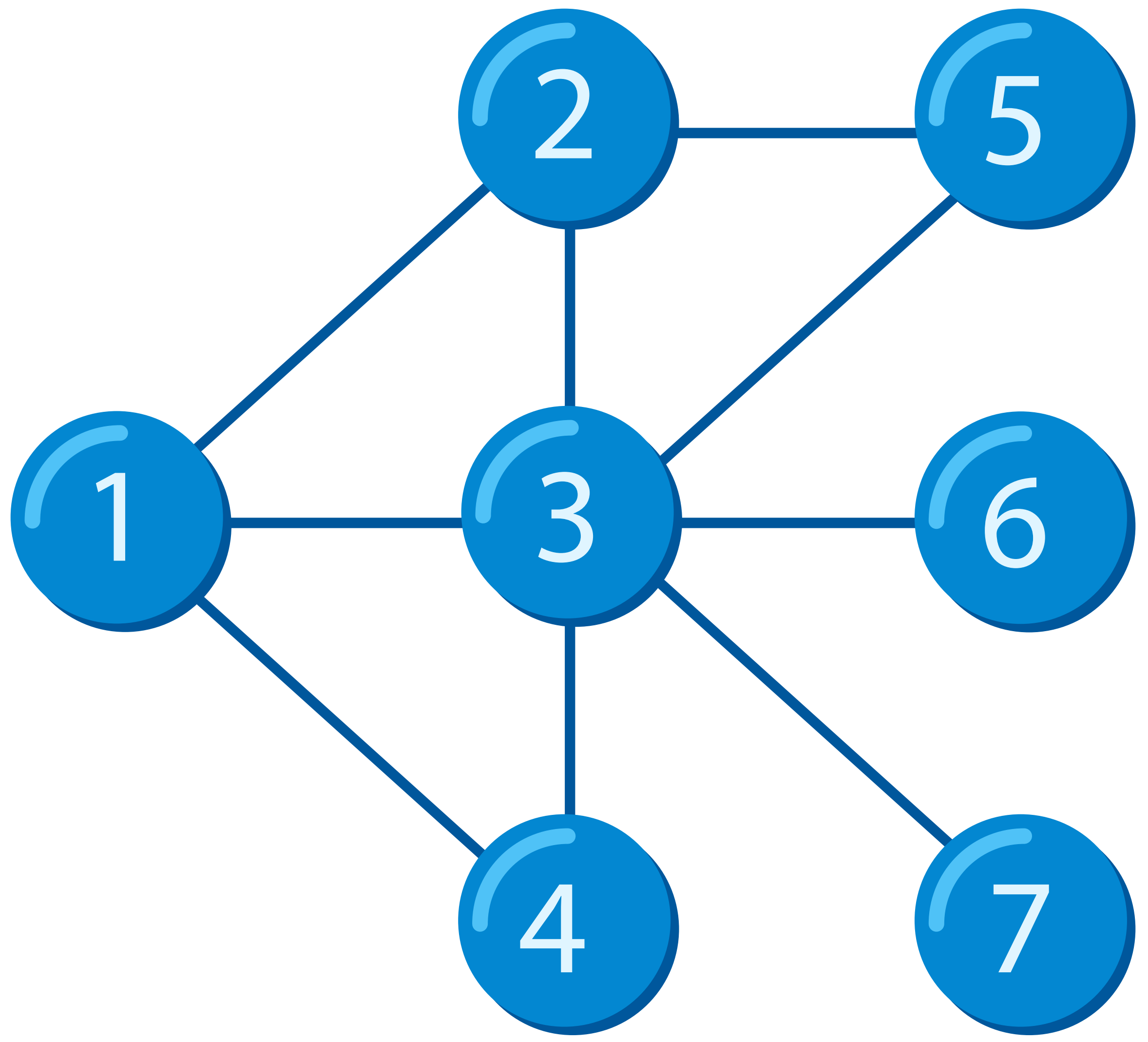
3 2

```
bcDF.show()
+---+---+
|  B|  C|
+---+---+
|  1|  2|
|  1|  3|
|  1|  4|
|  2|  3|
|  2|  5|
|  3|  4|
|  3|  5|
|  3|  6|
|  3|  7|
+---+---+
```

3 2

```
+---+---+---+---+
|  A|  B|  B|  C|
+---+---+---+---+
|  1|  3|  3|  4|
|  1|  3|  3|  5|
|  1|  3|  3|  6|
|  1|  3|  3|  7|
|  1|  2|  2|  3|
|  1|  2|  2|  5|
+---+---+---+---+
```

1 3 3 2



Result

```
+---+---+-------+
|  A|  C|count|
+---+---+-------+
|  1|  3|      1|
|  1|  4|      1|
|  1|  5|      2|
|  1|  6|      1|
|  1|  7|      1|
+---+---+-------+
   1   2       1
```

```
<1, 2>
<2, 1>
<1, 3>
<3, 1>
<1, 4>
<4, 1>
<2, 3>
<3, 2>
<2, 5>
<5, 2>
<3, 4>
<4, 3>
<3, 5>
<5, 3>
<3, 6>
<6, 3>
<3, 7>
<7, 3>
```

```
<1, 2>
<2, 1>
<1, 3>
<3, 1>
<1, 4>
<4, 1>
<2, 3>
<3, 2>
<2, 5>
<5, 2>
<3, 4>
<4, 3>
<3, 5>
<5, 3>
<3, 6>
<6, 3>
<3, 7>
<7, 3>
```

join on

```
<1, 2>
<2, 1>
<1, 3>
<3, 1>
<1, 4>
<4, 1>
<2, 3>
<3, 2>
<2, 5>
<5, 2>
<3, 4>
<4, 3>
<3, 5>
<5, 3>
<3, 6>
<6, 3>
<3, 7>
<7, 3>
```

```
<1, 2>
<2, 1>
<1, 3>
<3, 1>
<1, 4>
<4, 1>
<2, 3>
<3, 2>
<2, 5>
<5, 2>
<3, 4>
<4, 3>
<3, 5>
<5, 3>
<3, 6>
<6, 3>
<3, 7>
<7, 3>
```

join on

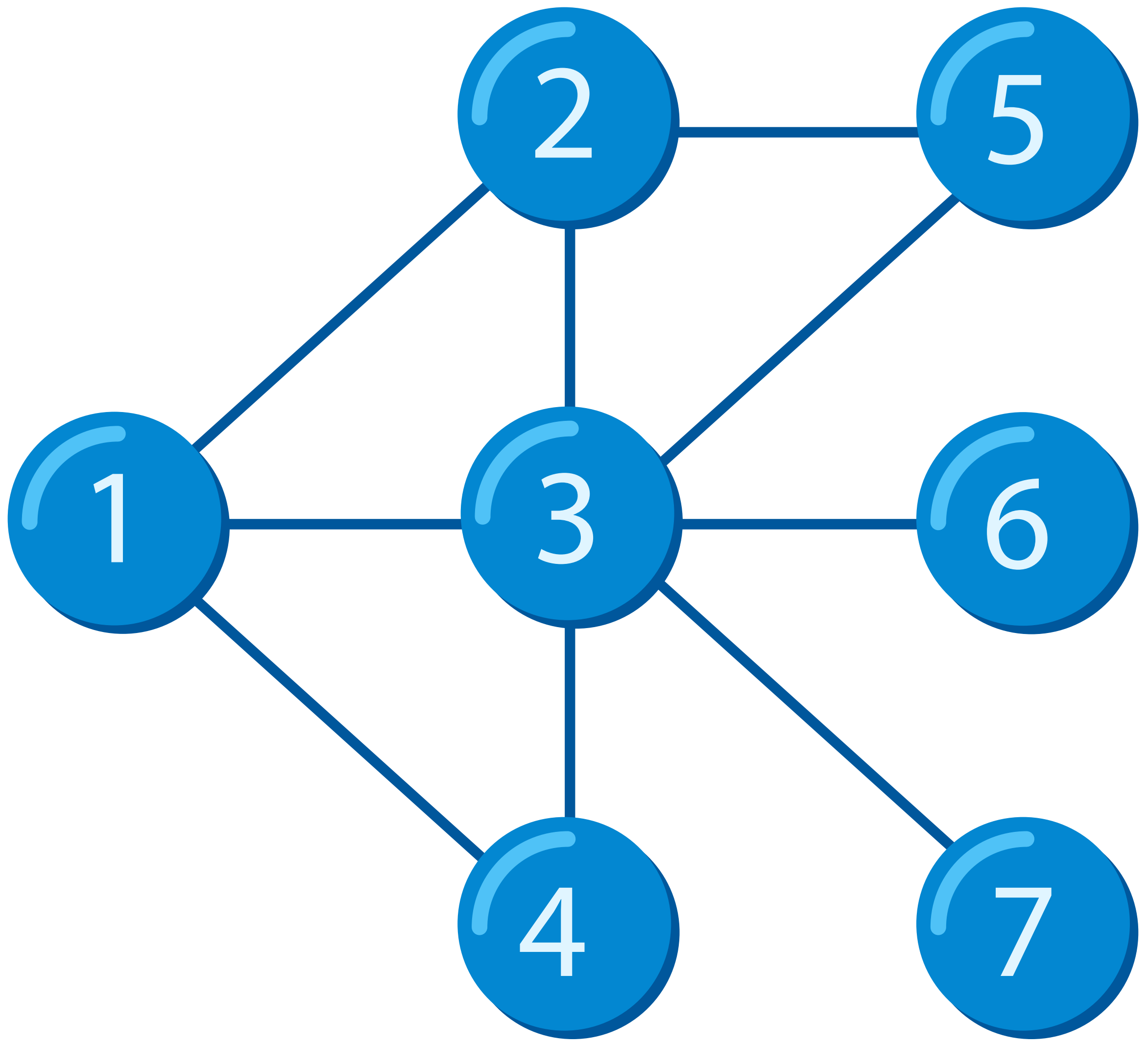
```
<1, 2>
<2, 1>
<1, 3>
<3, 1>
<1, 4>
<4, 1>
<2, 3>
<3, 2>
<2, 5>
<5, 2>
<3, 4>
<4, 3>
<3, 5>
<5, 3>
<3, 6>
<6, 3>
<3, 7>
<7, 3>
```

$\longrightarrow$

for user 1

```
<1,(2, 1)>
<1,(2, 3)>
<1,(2, 5)>
<1,(3, 1)>
<1,(3, 2)>
<1,(3, 4)>
<1,(3, 5)>
<1,(3, 6)>
<1,(3, 7)>
<1,(4, 1)>
<1,(4, 3)>
```

join for user 1	no "friend in the middle"	result
`<1,(2, 1)>`	`<1, 1>`	`<(1, 1), 3>`
`<1,(2, 3)>`	`<1, 3>`	`<(1, 2), 1>`
`<1,(2, 5)>`	`<1, 5>`	`<(1, 3), 2>`
`<1,(3, 1)>`	`<1, 1>`	`<(1, 4), 1>`
`<1,(3, 2)>`	`<1, 2>`	`<(1, 5), 2>`
`<1,(3, 4)>`	`<1, 4>`	`<(1, 6), 1>`
`<1,(3, 5)>`	`<1, 5>`	`<(1, 7), 1>`
`<1,(3, 6)>`	`<1, 6>`	
`<1,(3, 7)>`	`<1, 7>`	
`<1,(4, 1)>`	`<1, 1>`	
`<1,(4, 3)>`	`<1, 3>`	



Result

```
<(1, 1), 3>
<(1, 2), 1>
<(1, 3), 2>
<(1, 4), 1>
<(1, 5), 2>
<(1, 6), 1>
<(1, 7), 1>
```

Edge list algorithm:

1. For each edge emit its reversed copy
2. Join RDD from step 1 on itself
3. Throw away "friend in the middle"
4. For each pair count number of occurrences
5. Filter pairs consisting of identical vertices

Edge list algorithm:

1. For each edge emit its reversed copy
2. Join RDD from step 1 on itself
3. Throw away "friend in the middle"
4. For each pair count number of occurrences
5. Filter pairs consisting of identical vertices

double data

Edge list algorithm:

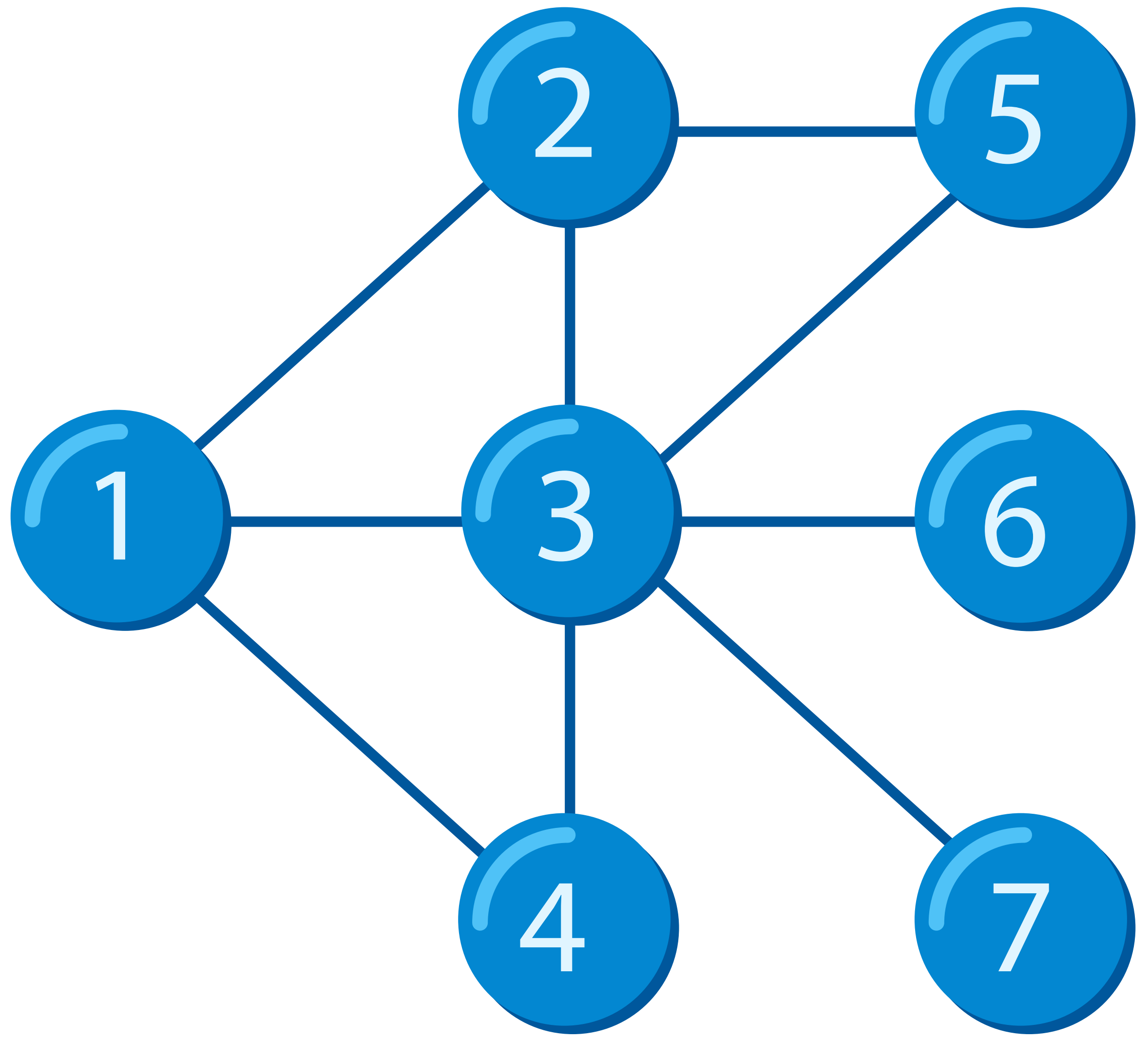
1. For each edge emit its reversed copy
2. Join RDD from step 1 on itself
3. Throw away "friend in the middle"
4. For each pair count number of occurrences
5. Filter pairs consisting of identical vertices

double data

shuffle

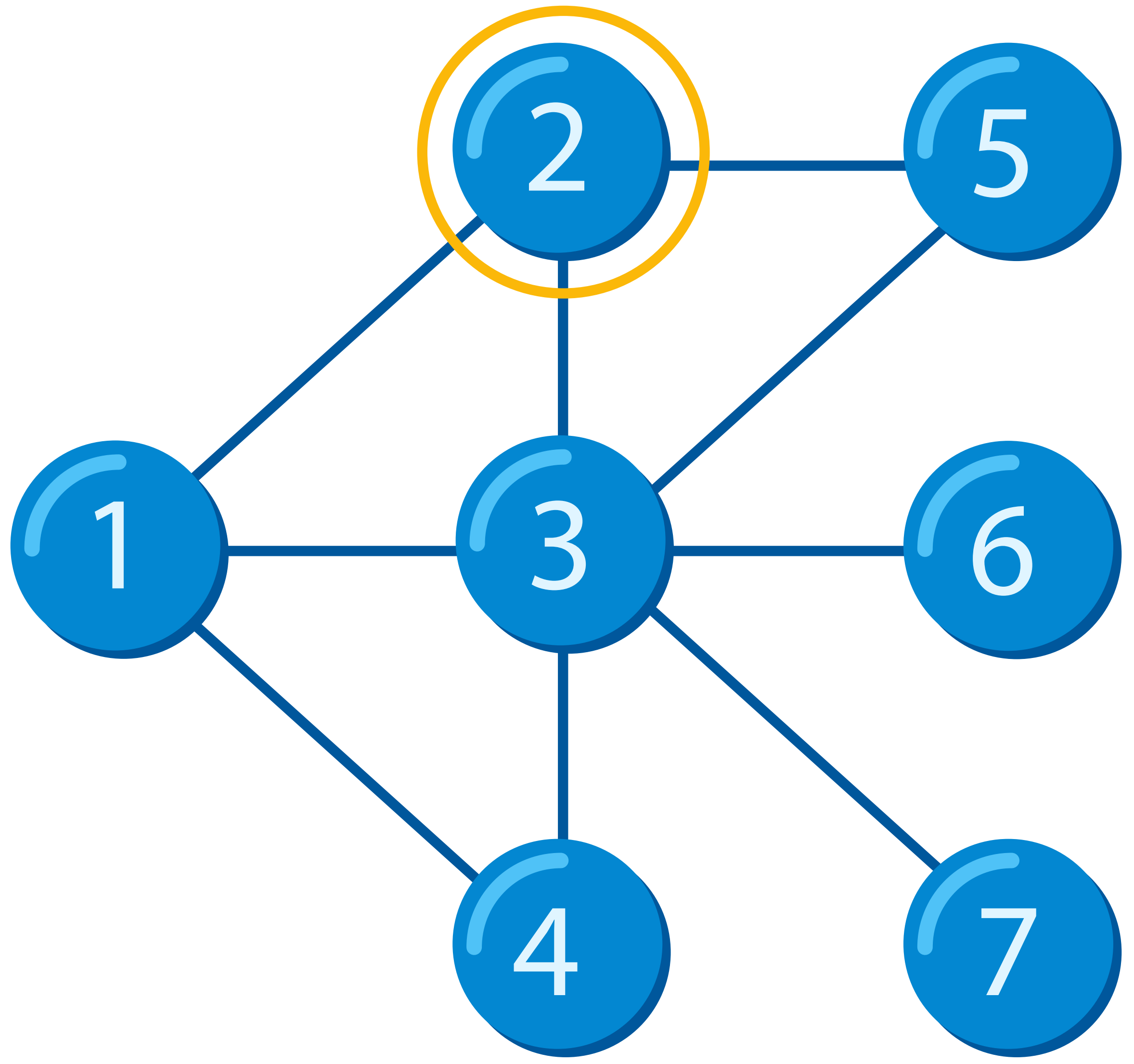
Edge list?

$\boxed{\text{Adjacency lists?}}$



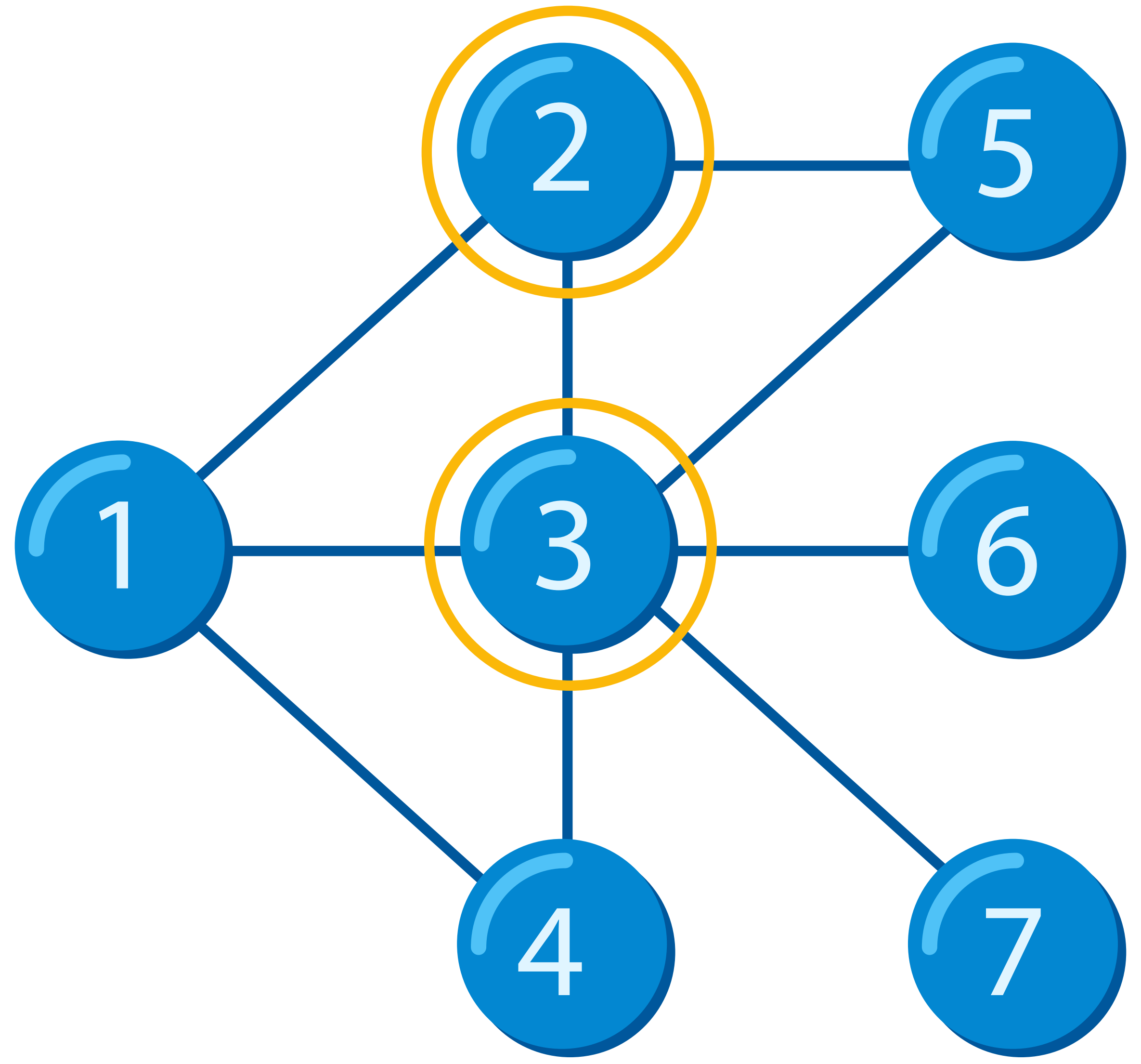
Adjacency lists

```
[[2,3,4],
[1,3,5],
[1, 2, 4, 5, 6 ,7],
[1, 3],
[2, 3],
[3],
[3]]
```



Adjacency lists

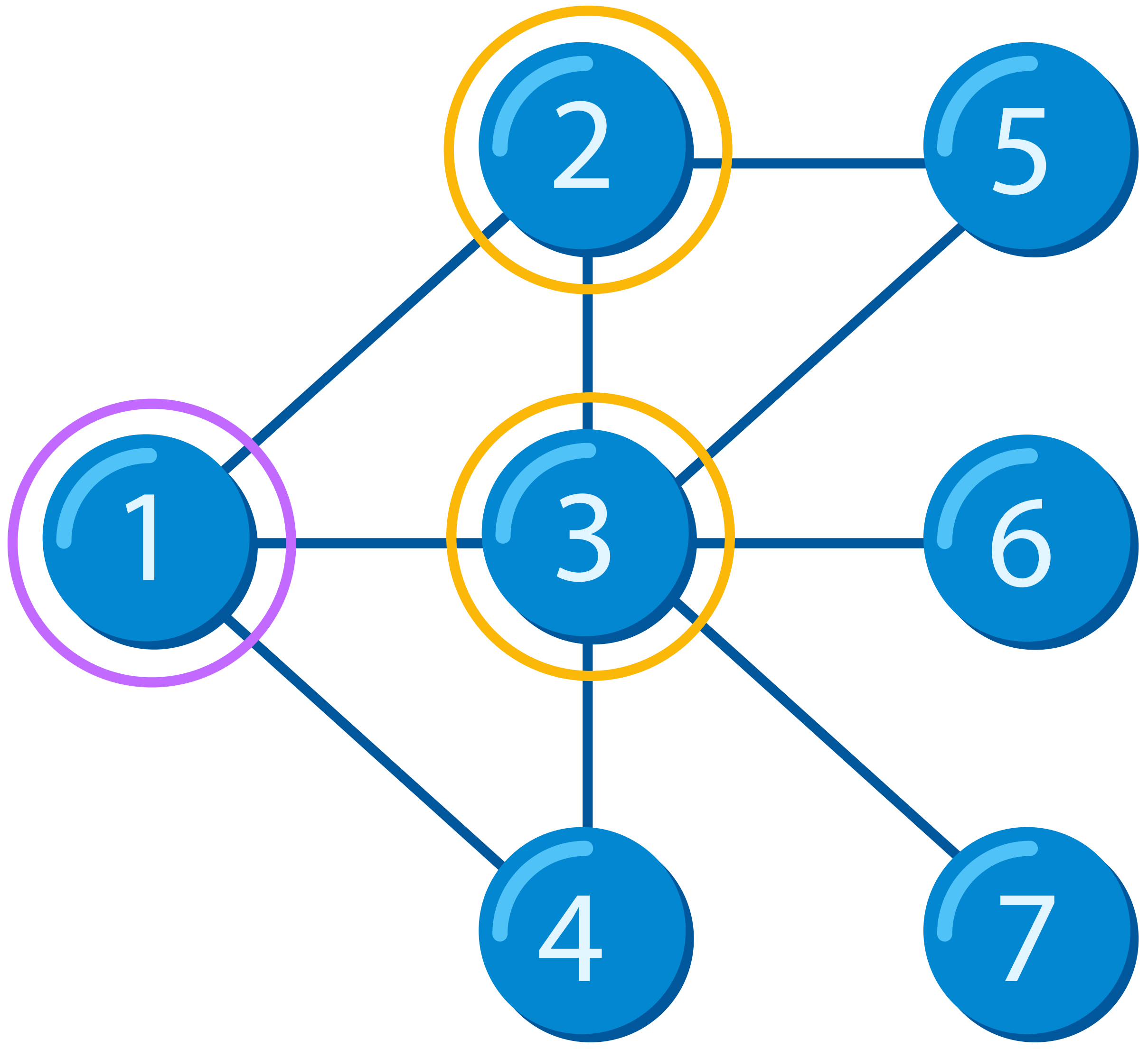
[[2,3,4],
[1,3,5],
[1, 2, 4, 5, 6 ,7],
[1, 3],
[2, 3],
[3],
[3]]



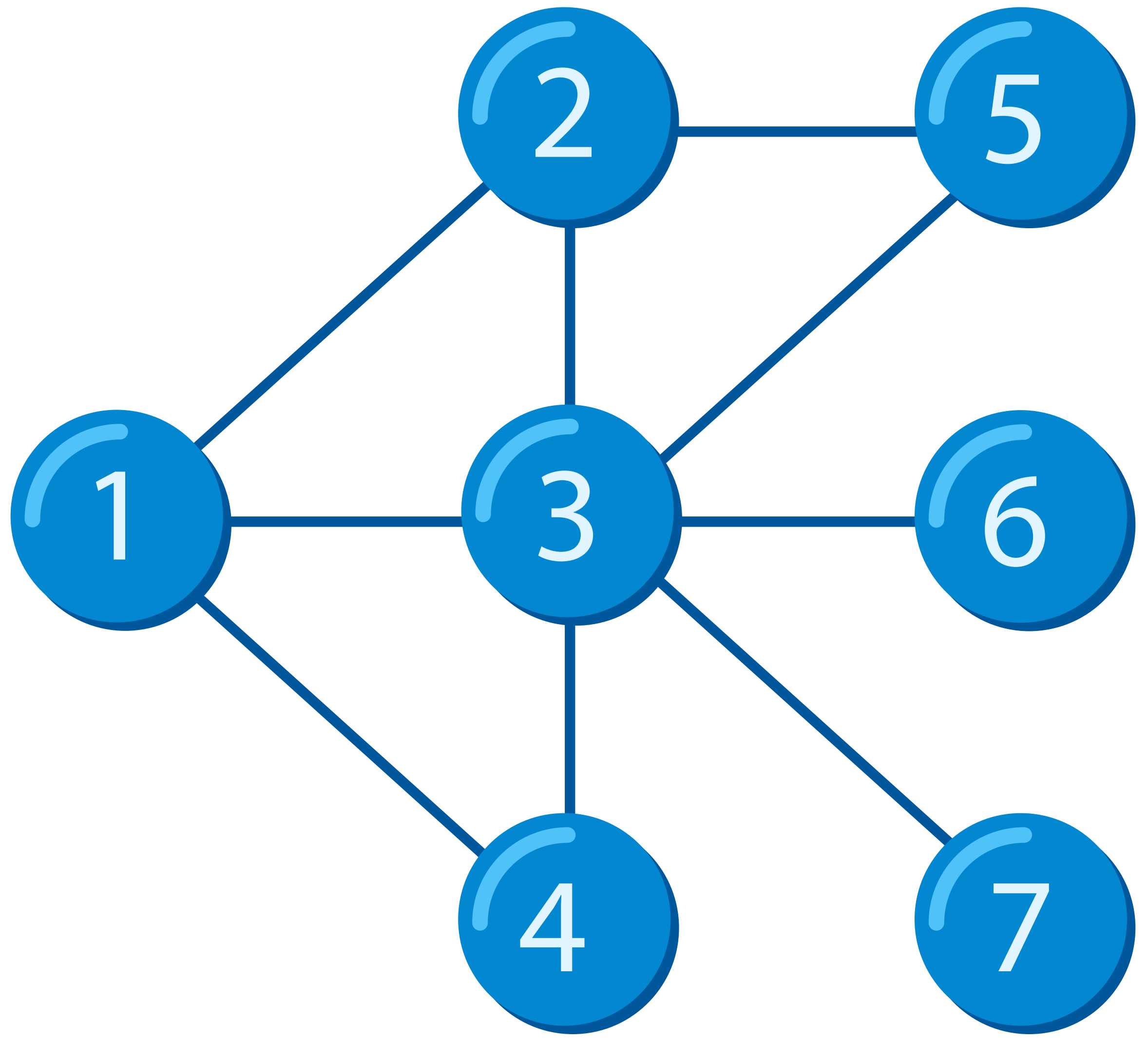
Adjacency lists

[[2,3,4],
[1,3,5],
[1, 2, 4, 5, 6 ,7],
[1, 3],
[2, 3],
[3],
[3]]

Adjacency lists

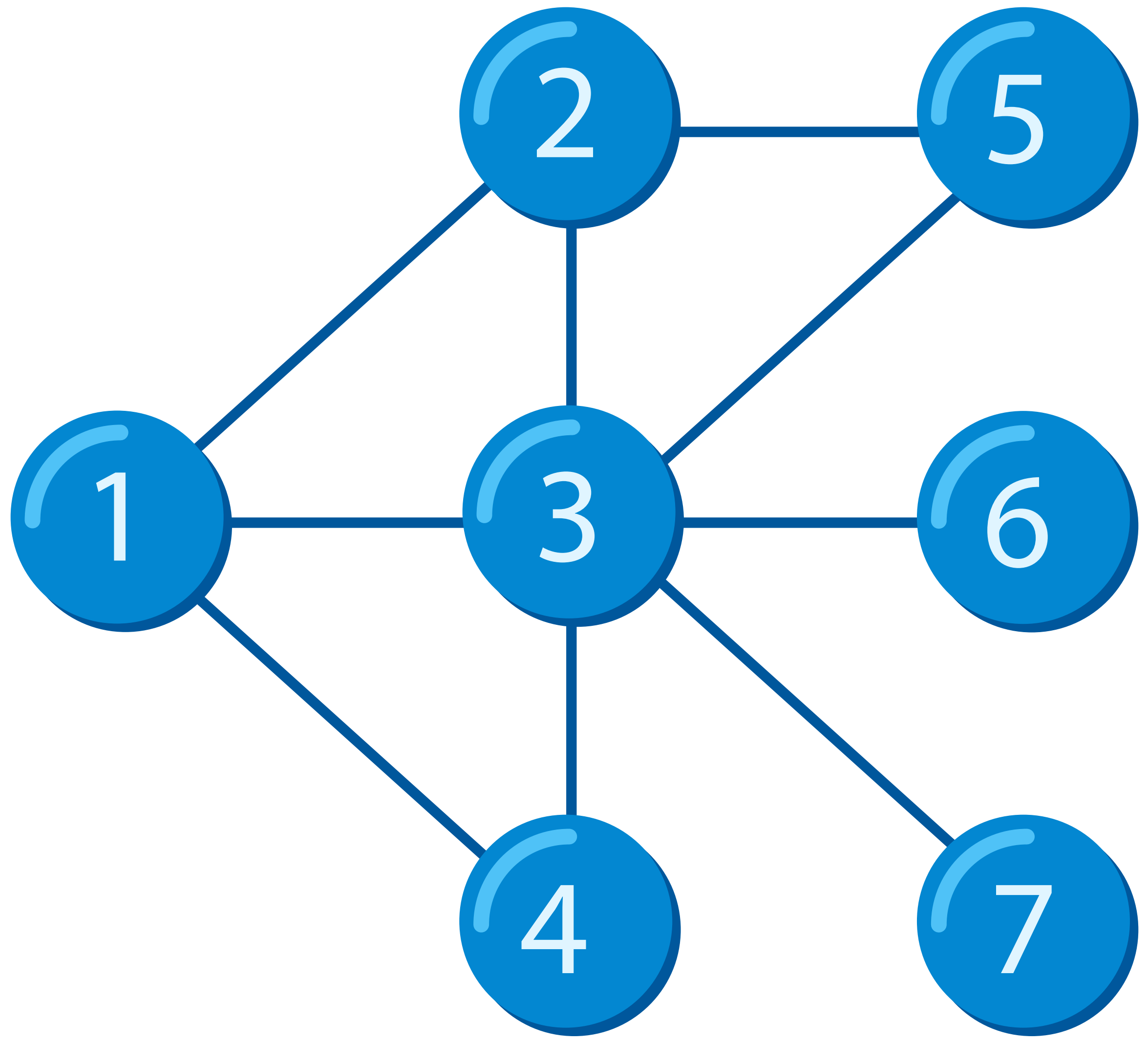


[[2,3,4],
[1,3,5],
[1, 2, 4, 5, 6 ,7],
[1, 3],
[2, 3],
[3],
[3]]



Adjacency lists

[[2,3,4],
[1,3,5],
[1, 2, 4, 5, 6 ,7],
[1, 3],
[2, 3],
[3],
[3]]



Adjacency lists

[[2,3,4],	<2, 3>
[1,3,5],	<3, 2>
[1, 2, 4, 5, 6 ,7],	<4, 2>
[1, 3],	<2, 4>
[2, 3],	<3, 4>
[3],	<4, 3>
[3]]	

Adjacency lists algorithm:

1. For each **adjacency list emit all possible pairs**
2. For each pair count number of occurrences

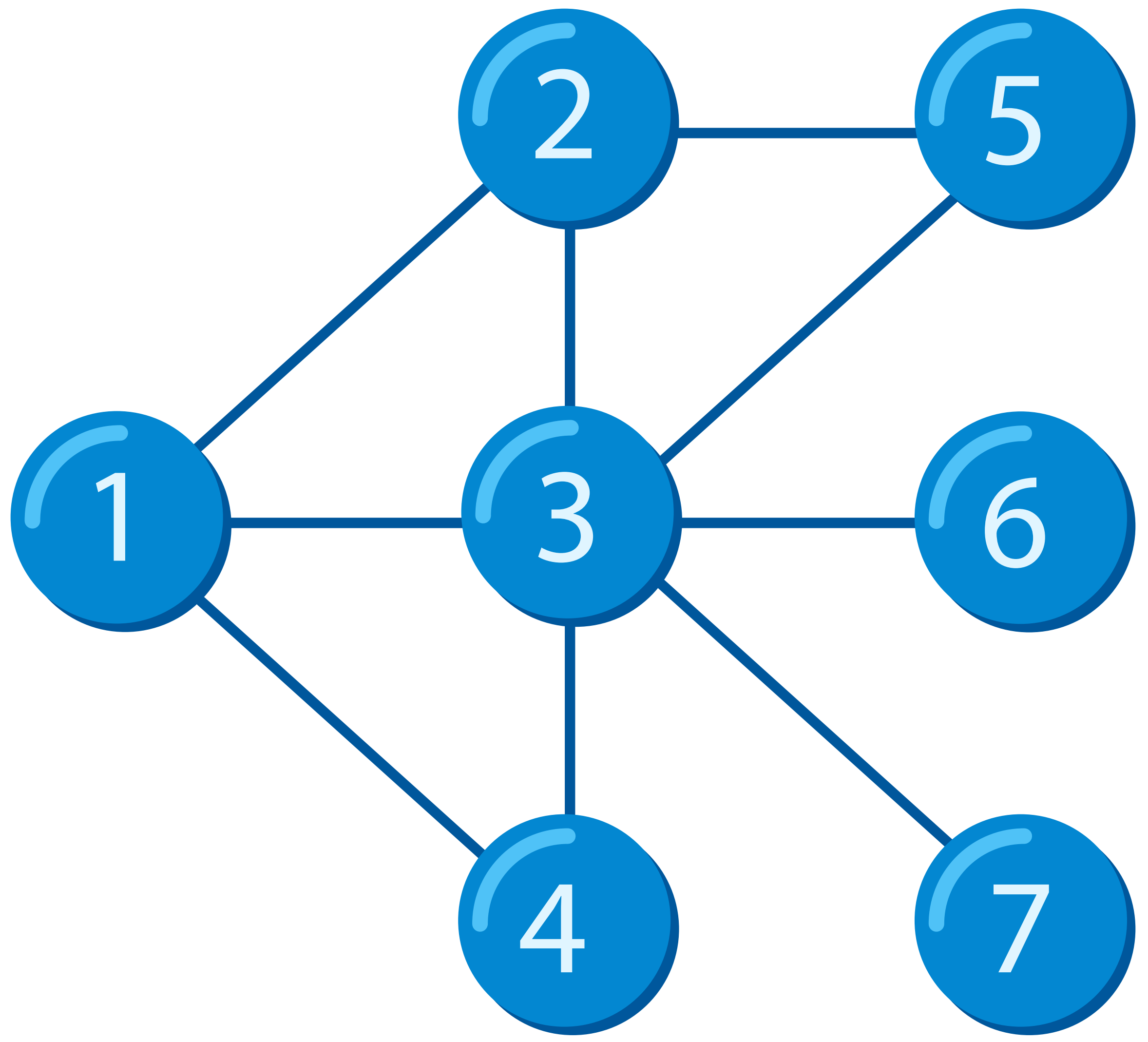
Adjacency lists algorithm:

1. For each **adjacency list emit all possible pairs**

2. For each pair count number of occurrences

data explode

Undirected



$$\begin{tikzcd}<(1, 3), 2> \arrow[d] \\ <(3, 1), 2>\end{tikzcd}$$

adjacency lists

```
[[2,3,4],
[1,3,5],
[1, 2, 4, 5, 6 ,7],
[1, 3],
[2, 3],
[3],
[3]]
```

all pairs involve user1

```
<1, 3>
<1, 5>
<1, 2>
<1, 4>
<1, 5>
<1, 6>
<1, 7>
<1, 3>
```

result

```
<(1, 2), 1>
<(1, 3), 2>
<(1, 4), 1>
<(1, 5), 2>
<(1, 6), 1>
<(1, 7), 1>
```

adjacency lists

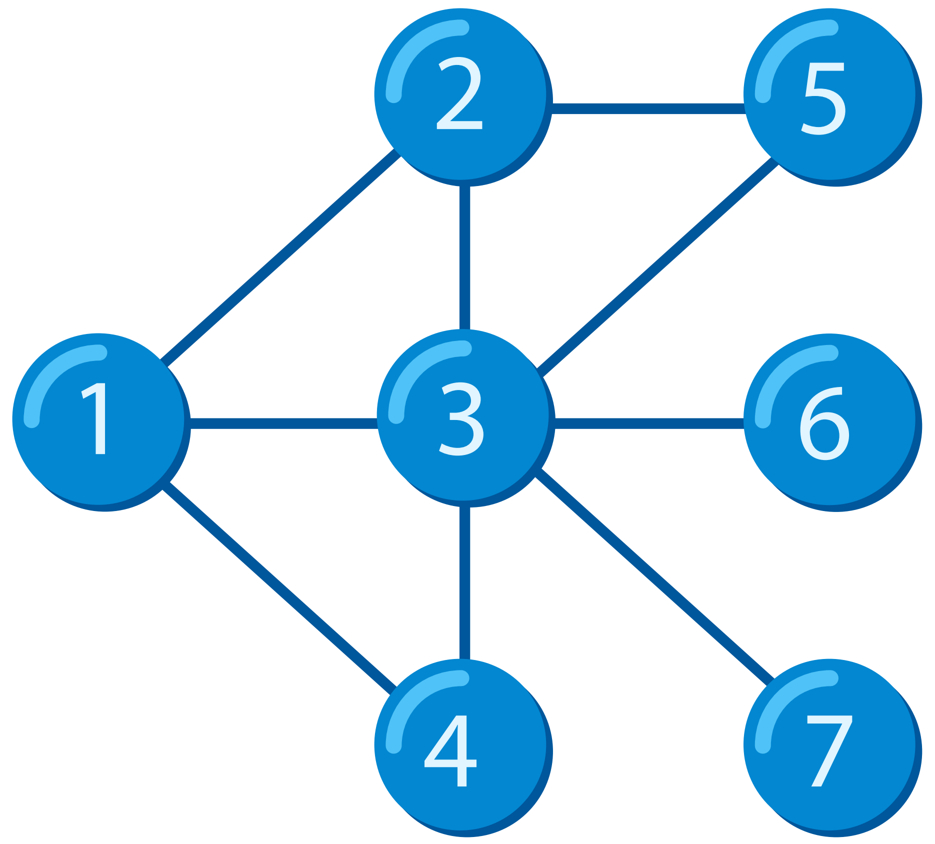
```
[[2,3,4],
[1,3,5],
[1, 2, 4, 5, 6 ,7],
[1, 3],
[2, 3],
[3],
[3]]
```

all pairs involve user1

```
<1, 3>
<1, 5>
<1, 2>
<1, 4>
<1, 5>
<1, 6>
<1, 7>
<1, 3>
```

result

```
<(1, 2), 1>
<(1, 3), 2>
<(1, 4), 1>
<(1, 5), 2>
<(1, 6), 1>
<(1, 7), 1>
```



modified adjacency lists algorithm:

1. Sort all adjacency lists in ascending order
2. For each **adjacency list emit all ordered pairs**
3. For each pair count number of occurrences

adjacency lists algorithm:

1. For each **adjacency list emit all possible pairs**

2. For each pair count number of occurrences

modified adjacency lists algorithm:

1. Sort all adjacency lists in ascending order

2. For each **adjacency list emit all ordered pairs**

3. For each pair count number of occurrences

for user 1

```
<1,(2, 1)> <(2, 1),1>
<1,(3, 2)> <(3, 1),2>
<1,(4, 1)> <(4, 1),1>
<1,(5, 2)> <(5, 1),2>
<1,(6, 1)> <(6, 1),1>
<1,(7, 1)> <(7, 1),1>
```

for user 1

```
<(1, 2), 1>
<(1, 3), 2>
<(1, 4), 1>
<(1, 5), 2>
<(1, 6), 1>
<(1, 7), 1>
```

Summary

- Now you know that selection of graph representation type can critically influence the effectiveness of further graph processing.

Summary

- Now you know that selection of graph representation type can critically influence the effectiveness of further graph processing.
- So before making choice between adjacency matrix, edge lists and adjacency lists you should have clear understanding of how you will proceed data further.