Spark - Exercises

Exercise #30

- Log filtering
  - Input: a simplified log of a web server (i.e., a textual file)
    - Each line of the file is associated with a URL request
  - Output: the lines containing the word "google"
    - Store the output in an HDFS folder

Exercise #30 - Example

- Input file

```
66.249.69.97 - - [24/Sep/2014:22:25:44 +0000] "GET http://www.google.com/bot.html”
66.249.69.97 - - [24/Sep/2014:22:26:44 +0000] "GET http://www.google.com/how.html”
66.249.69.97 - - [24/Sep/2014:22:28:44 +0000] "GET http://dbdmg.polito.it/course.html”
71.19.157.179 - - [24/Sep/2014:22:30:12 +0000] "GET http://www.google.com/faq.html”
66.249.69.97 - - [24/Sep/2014:31:28:44 +0000] "GET http://dbdmg.polito.it/thesis.html”
```

- Output

```
66.249.69.97 - - [24/Sep/2014:22:25:44 +0000] "GET http://www.google.com/bot.html”
66.249.69.97 - - [24/Sep/2014:22:26:44 +0000] "GET http://www.google.com/how.html”
71.19.157.179 - - [24/Sep/2014:22:30:12 +0000] "GET http://www.google.com/faq.html”
```

Exercise #31

- Log analysis
  - Input: log of a web server (i.e., a textual file)
    - Each line of the file is associated with a URL request
  - Output: the list of distinct IP addresses associated with the connections to a google page (i.e., connections to URLs containing the term "www.google.com")
    - Store the output in an HDFS folder

Exercise #31 - Example

- Input file

```
66.249.69.97 - - [24/Sep/2014:22:25:44 +0000] "GET http://www.google.com/bot.html"
66.249.69.97 - - [24/Sep/2014:22:26:44 +0000] "GET http://www.google.com/how.html"
66.249.69.97 - - [24/Sep/2014:22:28:44 +0000] "GET http://dbdmg.polito.it/course.html"
71.19.157.179 - - [24/Sep/2014:22:30:12 +0000] "GET http://www.google.com/faq.html"
66.249.69.95 - - [24/Sep/2014:31:28:44 +0000] "GET http://dbdmg.polito.it/thesis.html"
66.249.69.97 - - [24/Sep/2014:56:26:44 +0000] "GET http://www.google.com/how.html"
56.249.69.97 - - [24/Sep/2014:56:26:44 +0000] "GET http://www.google.com/how.html"
```

- Output

```
66.249.69.97
71.19.157.179
56.249.69.97
```

Exercise #32

- Maximum value
  - Input: a collection of (structured) textual csv files containing the daily value of PM10 for a set of sensors
    - Each line of the files has the following format

      sensorId**,**date**,**PM10 value (µg/m^3)**\n**
  - Output: report the maximum value of PM10
    - Print the result on the standard output

Exercise #32 - Example

- Input file

```
s1,2016-01-01,20.5
s2,2016-01-01,30.1
s1,2016-01-02,60.2
s2,2016-01-02,20.4
s1,2016-01-03,55.5
s2,2016-01-03,52.5
```

- Output

60.2

Exercise #33

- Top-k maximum values
  - Input: a collection of (structured) textual csv files containing the daily value of PM10 for a set of sensors
    - Each line of the files has the following format

      sensorId**,**date**,**PM10 value (μg/m^3)**\n**
  - Output: report the top-3 maximum values of PM10
    - Print the result on the standard output

Exercise #33 - Example

- Input file

```
s1,2016-01-01,20.5
s2,2016-01-01,30.1
s1,2016-01-02,60.2
s2,2016-01-02,20.4
s1,2016-01-03,55.5
s2,2016-01-03,52.5
```

- Output

```
60.2
55.5
52.5
```

Exercise #34

- Readings associated with the maximum value
  - Input: a collection of (structured) textual csv files containing the daily value of PM10 for a set of sensors
    - Each line of the files has the following format

      sensorId**,**date**,**PM10 value (μg/m^3)**\n**
  - Output: the line(s) associated with the maximum value of PM10
    - Store the result in an HDFS folder

Exercise #34 - Example

- Input file

```
s1,2016-01-01,20.5
s2,2016-01-01,30.1
s1,2016-01-02,60.2
s2,2016-01-02,20.4
s1,2016-01-03,60.2
s2,2016-01-03,52.5
```

- Output

```
s1,2016-01-02,60.2
s1,2016-01-03,60.2
```

Exercise #35

- Dates associated with the maximum value
  - Input: a collection of (structured) textual csv files containing the daily value of PM10 for a set of sensors
    - Each line of the files has the following format

      sensorId**,**date**,**PM10 value (μg/m^3)**\n**
  - Output: the date(s) associated with the maximum value of PM10
    - Store the result in an HDFS folder

Exercise #35 - Example

- Input file

```
s1,2016-01-01,20.5
s2,2016-01-01,30.1
s1,2016-01-02,60.2
s2,2016-01-02,20.4
s1,2016-01-03,60.2
s2,2016-01-03,52.5
```

- Output

```
2016-01-02
2016-01-03
```

Exercise #36

- Average value
  - Input: a collection of (structured) textual csv files containing the daily value of PM10 for a set of sensors
    - Each line of the files has the following format

      sensorId**,**date**,**PM10 value (μg/m^3)**\n**
  - Output: compute the average PM10 value
    - Print the result on the standard output

Exercise #36 - Example

- Input file

```
s1,2016-01-01,20.5
s2,2016-01-01,30.1
s1,2016-01-02,60.2
s2,2016-01-02,20.4
s1,2016-01-03,55.5
s2,2016-01-03,52.5
```

- Output

39.86

Exercise #37

- Maximum values
  - Input: a textual csv file containing the daily value of PM10 for a set of sensors
    - Each line of the files has the following format

      sensorId**,**date**,**PM10 value (μg/m^3)**\n**
  - Output: the maximum value of PM10 for each sensor
    - Store the result in an HDFS file

Exercise #37 - Example

- Input file

```
s1,2016-01-01,20.5
s2,2016-01-01,30.1
s1,2016-01-02,60.2
s2,2016-01-02,20.4
s1,2016-01-03,55.5
s2,2016-01-03,52.5
```

- Output

```
(s1,60.2)
(s2,52.5)
```

Exercise #38

- Pollution analysis
  - Input: a textual csv file containing the daily value of PM10 for a set of sensors
    - Each line of the files has the following format

      sensorId**,**date**,**PM10 value (μg/m^3)**\n**
  - Output: the sensors with at least 2 readings with a PM10 value greater than the critical threshold 50
    - Store in an HDFS file the sensorIds of the selected sensors and also the number of times each of those sensors is associated with a PM10 value greater than 50

Exercise #38 - Example

- Input file

```
s1,2016-01-01,20.5
s2,2016-01-01,30.1
s1,2016-01-02,60.2
s2,2016-01-02,20.4
s1,2016-01-03,55.5
s2,2016-01-03,52.5
```

- Output

(s1,2)

Exercise #39

- Critical dates analysis
  - Input: a textual csv file containing the daily value of PM10 for a set of sensors
    - Each line of the files has the following format

      sensorId**,**date**,**PM10 value (μg/m^3)**\n**
  - Output: an HDFS file containing one line for each sensor
    - Each line contains a sensorId and the list of dates with a PM10 values greater than 50 for that sensor
  - **Consider only the sensors associated at least one time with a PM10 value greater than 50**

Exercise #39 - Example

- Input file

```
s1,2016-01-01,20.5
s2,2016-01-01,30.1
s1,2016-01-02,60.2
s2,2016-01-02,20.4
s1,2016-01-03,55.5
s2,2016-01-03,52.5
```

- Output

(s1, [2016-01-02, 2016-01-03])
(s2, [2016-01-03])

Exercise #39 bis

- Critical dates analysis
  - Input: a textual csv file containing the daily value of PM10 for a set of sensors
    - Each line of the files has the following format

      sensorId**,**date**,**PM10 value (µg/m^3)**\n**
  - Output: an HDFS file containing one line for each sensor
    - Each line contains a sensorId and the list of dates with a PM10 values greater than 50 for that sensor
    - Also the sensors which have never been associated with a PM10 values greater than 50 must be included in the result (with an empty set)

Exercise #39 bis - Example

- Input file

```
s1,2016-01-01,20.5
s2,2016-01-01,30.1
s1,2016-01-02,60.2
s2,2016-01-02,20.4
s1,2016-01-03,55.5
s2,2016-01-03,52.5
s3,2016-01-03,12.5
```

- Output

(s1, [2016-01-02, 2016-01-03])
(s2, [2016-01-03])
(s3, [])

Exercise #40

- Order sensors by number of critical days
  - Input: a textual csv file containing the daily value of PM10 for a set of sensors
    - Each line of the files has the following format

      sensorId**,**date**,**PM10 value (μg/m^3)**\n**
  - Output: an HDFS file containing the sensors ordered by the number of critical days
    - Each line of the output file contains the number of days with a PM10 values greater than 50 for a sensor **s** and the sensorId of sensor **s**
  - **Consider only the sensors associated at least one time with a PM10 value greater than 50**

Exercise #40 - Example

- Input file

```
s1,2016-01-01,20.5
s2,2016-01-01,30.1
s1,2016-01-02,60.2
s2,2016-01-02,20.4
s1,2016-01-03,55.5
s2,2016-01-03,52.5
```

- Output

```
2, s1
1, s2
```

Exercise #41

- Top-k most critical sensors
  - Input:
    - A textual csv file containing the daily value of PM10 for a set of sensors
      - Each line of the files has the following format

        sensorId**,**date**,**PM10 value (µg/m^3)**\n**
    - The value of k
      - It is an argument of the application

Exercise #41

- Top-k most critical sensors
  - Output:
    - An HDFS file containing the top-k critical sensors
      - The “criticality” of a sensor is given by the number of days with a PM10 values greater than 50
      - Each line contains the number of critical days and the sensorId

Exercise #41 - Example

- Input file

```
s1,2016-01-01,20.5
s2,2016-01-01,30.1
s1,2016-01-02,60.2
s2,2016-01-02,20.4
s1,2016-01-03,55.5
s2,2016-01-03,52.5
```

  - k = 1
- Output

2, s1

Exercise #42

- Mapping Question-Answer(s)
  - Input:
    - A large textual file containing a set of questions
      - Each line contains one question
      - Each line has the format
        - QuestionId,Timestamp,TextOfTheQuestion
    - A large textual file containing a set of answers
      - Each line contains one answer
      - Each line has the format
        - AnswerId,QuestionId,Timestamp,TextOfTheAnswer

Exercise #42

- Output:
  - A file containing one line for each question
  - Each line contains a question and the list of answers to that question
    - QuestionId, TextOfTheQuestion, list of Answers

Exercise #42 - Example

- Questions

```
Q1,2015-01-01,What is ..?
Q2,2015-01-03,Who invented ..
```

- Answers

```
A1,Q1,2015-01-02,It is ..
A2,Q2,2015-01-03,John Smith
A3,Q1,2015-01-05,I think it is ..
```

Exercise #42 - Example

- Output

(Q1,([What is ..?],[It is .., I think it is ..]))
(Q2,([Who invented ..],[John Smith]))

Exercise #43 – 1

- Critical bike sharing station analysis
- Input:
  - A textual csv file containing the occupancy of the stations of a bike sharing system
    - The sampling rate is 5 minutes
    - Each line of the file contains one sensor reading/sample has the following format

      stationId,date,hour,minute,num_of_bikes,num_of_free_slots
    - Some readings are missing due to temporarily malfunctions of the stations
      - Hence, the number of samplings is not exactly the same for all stations
  - The number of distinct stations is 100

Exercise #43 – 2

- Input:
  - A second textual csv file containing the list of neighbors of each station
    - Each line of the file has the following format
      stationId$_x$, list of neighbors of stationId$_x$
    - E.g.,
      s1,s2 s3
      means that s2 and s3 are neighbors of s1

Exercise #43 – 3

- Outputs:
  - Compute the percentage of critical situations for each station
    - A station is in a critical situation if the number of free slots is below a user provided threshold (e.g., 3 slots)
    - The percentage of critical situations for a station Si is defined as (number of critical readings associated with Si)/(total number of readings associated with Si)

Exercise #43 – 4

- Store in an HDFS file the stations with a percentage of critical situations higher than 80% (i.e., stations that are almost always in a critical situation and need to be extended)
  - Each line of the output file is associated with one of the selected stations and contains the percentage of critical situations and the stationId
  - Sort the stored stations by percentage of critical situations

Exercise #43 – 5

- Compute the percentage of critical situations for each pair (timeslot, station)
  - Timeslot can assume the following 6 values
    - [0-3]
    - [4-7]
    - [8-11]
    - [12-15]
    - [16-19]
    - [20-23]

Exercise #43 – 6

- Store in an HDFS file the pairs (timeslot, station) with a percentage of critical situations higher than 80% (i.e., stations that need rebalancing operations in specific timeslots)
  - Each line of the output file is associated with one of the selected pairs (timeslot, station) and contains the percentage of critical situations and the pair (timeslot, stationId)
  - Sort the result by percentage of critical situations

Exercise #43 – 7

- Select a reading (i.e., a line) of the first input file if and only if the following constraints are true
  - The line is associated with a full station situation
    - i.e., the station Si associated with the current line has a number of free slots equal to 0
  - All the neighbor stations of the station Si are full in the time stamp associated with the current line
    - i.e., bikers cannot leave the bike at Station Si and also all the neighbor stations are full in the same time stamp
- Store the selected readings/lines in an HDFS file and print on the standard output the total number of such lines

Exercise #44

- Misleading profile selection
- Input:
  - A textual file containing the list of movies watched by the users of a video on demand service
    - Each line of the file contains the information about one visualization

      userid,movieid,start-timestamp,end-timestamp
    - The user with id *userid* watched the movie with id *movieid* from *start-timestamp* to *end-timestamp*

Exercise #44

- Input:
  - A second textual file containing the list of preferences for each user
    - Each line of the file contains the information about one preference

      userid,movie-genre
    - The user with id *userid* liked the movie of type *movie-genre*

Exercise #44

- Input:
  - A third textual file containing the list of movies with the associated information
    - Each line of the file contains the information about one movie

      movieid,title,movie-genre
    - There is only one line for each movie
      - i.e., each movie has one single genre

Exercise #44

- Output:
  - Select the userids of the list of users with a misleading profile
    - A user has a misleading profile if more than **threshold%** of the movies he/she watched are not associated with a movie genre he/she likes
    - **threshold** is an argument/parameter of the application and it is specified by the user
  - Store the result in an HDFS file

Exercise #45

- Profile update
- Input:
  - A textual file containing the list of movies watched by the users of a video on demand service
    - Each line of the file contains the information about one visualization

      userid,movieid,start-timestamp,end-timestamp
    - The user with id *userid* watched the movie with id *movieid* from *start-timestamp* to *end-timestamp*

Exercise #45

- Input:
  - A second textual file containing the list of preferences for each user
    - Each line of the file contains the information about one preference

      userid,movie-genre
    - The user with id *userid* liked the movie of type *movie-genre*

Exercise #45

- Input:
  - A third textual file containing the list of movies with the associated information
    - Each line of the file contains the information about one movie

      movieid,title,movie-genre
    - There is only one line for each movie
      - i.e., each movie has one single genre

Exercise #45

- Output:
  - Select for each user with a misleading profile (according to the same definition of Exercise #44) the list of movie genres that are not in his/her preferred genres and are associated with at least 5 movies watched by the user
  - Store the result in an HDFS file
    - Each line of the output file is associated with one pair (user, selected misleading genre) associated with him/her
    - The format is

      userid, selected (misleading) genre
    - Users associated with a list of selected genres are associated with multiple lines of the output file

Exercise #46

- Time series analysis
- Input:
  - A textual file containing a set of temperature readings
  - Each line of the file contains one timestamp and the associated temperature reading

    timestamp, temperature

    - The format of the timestamp is the Unix timestamp that is defined as the number of seconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC), Thursday, 1 January 1970
  - The sample rate is 1 minute
    - i.e., the difference between the timestamps of two consecutive readings is 60 seconds

Exercise #46

- Output:
  - Consider all the windows containing 3 consecutive temperature readings and
    - Select the windows characterized by an increasing trend
      - A window is characterized by an increasing trend if for all the temperature readings in it temperature(t)>temperature(t-60 seconds)
    - Store the result into an HDFS file

Exercise #46 - Example

- Input file

```
1451606400,12.1
1451606460,12.2
1451606520,13.5
1451606580,14.0
1451606640,14.0
1451606700,15.5
1451606760,15.0
```

- Output file

```
1451606400,12.1,1451606460,12.2,1451606520,13.5
1451606460,12.2,1451606520,13.5,1451606580,14.0
```

Exercise #47

- Input:
  - A CSV file containing a list of user profiles
    - Header
      - name,age,gender
    - Each line of the file contains the information about one user
- Output:
  - Select male users (gender="male"), increase by one their age, and store in the output folder name and age of these users sorted by decreasing age and ascending name (if the age value is the same)
  - The output does not contain the header line

Exercise #47

- Example of input data:

  name,age,gender

  Paul,40,male

  John,40,male

  David,15,male

  Susan,40,female

  Karen,34,female

- Example of expected output:

  John,41

  Paul,41

  David,16

Exercise #47

- Implement two different solutions for this exercise
  - A solution based only on DataFrames
  - A solution based on SQL like queries executed on a temporary table associated with the input data

Exercise #48

- Input:
  - A CSV file containing a list of user profiles
    - Header
      - name,age,gender
    - Each line of the file contains the information about one user
- Output:
  - Select the names occurring at least two times and store in the output folder name and average(age) of the selected names
  - The output does not contain the header line

Exercise #48

- Example of input data:

  name,age,gender

  Paul,40,male

  Paul,38,male

  David,15,male

  Susan,40,female

  Susan,34,female

- Example of expected output:

  Paul,39

  Susan,37

Exercise #48

- Implement two different solutions for this exercise
  - A solution based only on DataFrames
  - A solution based on SQL like queries executed on a temporary table associated with the input data

Exercise #49

- Input:
  - A csv file containing a list of profiles
    - Header: name,surname,age
    - Each line of the file contains one profile
    - name,surname,age
- Output:
  - A csv file containing one line for each profile. The original age attribute is substituted with a new attributed called rangeage of type String
    - rangeage = "[" + (age/10)*10 + "-" + (age/10)*10 +9"]"

Exercise #49

- Input:

  name,surname,age

  Paolo,Garza,42

  Luca,Boccia,41

  Maura,Bianchi,16

- Expected output:

  name,surname,rangeage

  Paolo,Garza,[40-49]

  Luca,Boccia,[40-49]

  Maura,Bianchi,[10-19]

Exercise #50

- Input:
  - A csv file containing a list of profiles
    - Header: name,surname,age
    - Each line of the file contains one profile
    - name,surname,age
- Output:
  - A csv file containing one single column called “name_surname” of type String
    - name_surname = name+" "+surname

Exercise #50

- Input:

  name,surname,age

  Paolo,Garza,42

  Luca,Boccia,41

  Maura,Bianchi,16

- Expected output:

  name_surname

  Paolo Garza

  Luca Boccia

  Maura Bianchi

Exercise #52

- GraphFrame
- Input:
  - The textual file vertexes.csv
    - It contains the vertexes of a graph
  - Each vertex is characterized by
    - id (string): user identifier
    - name (string): user name
    - age (integer): user age

Exercise #52

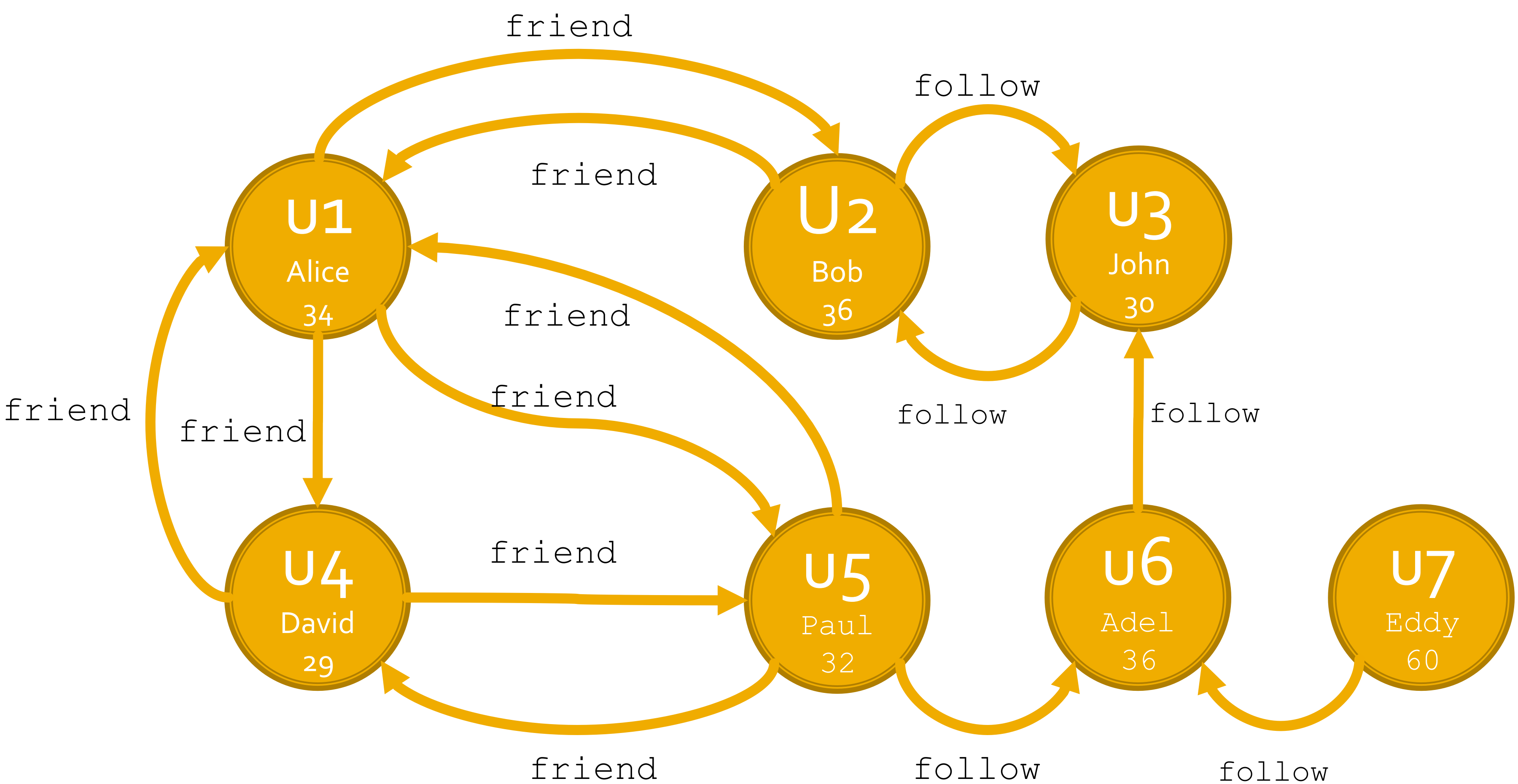
- The textual file edges.csv
  - It contains the edges of a graph
- Each edge is characterized by
  - src (string): source vertex
  - dst (string): destination vertex
  - linktype (string): "follow"or "friend"

Exercise #52

- Output:
  - For each user with at least one follower, store in the output folder the number of followers
    - One user per line
    - Format: user id, number of followers
  - Use the CSV format to store the result

Exercise #52

Input graph example



Exercise #52

- Result

id	numFollowers
u3	2
u6	2
u2	1

Exercise #53

- GraphFrame
- Input:
  - The textual file vertexes.csv
    - It contains the vertexes of a graph
  - Each vertex is characterized by
    - id (string): user identifier
    - name (string): user name
    - age (integer): user age

Exercise #53

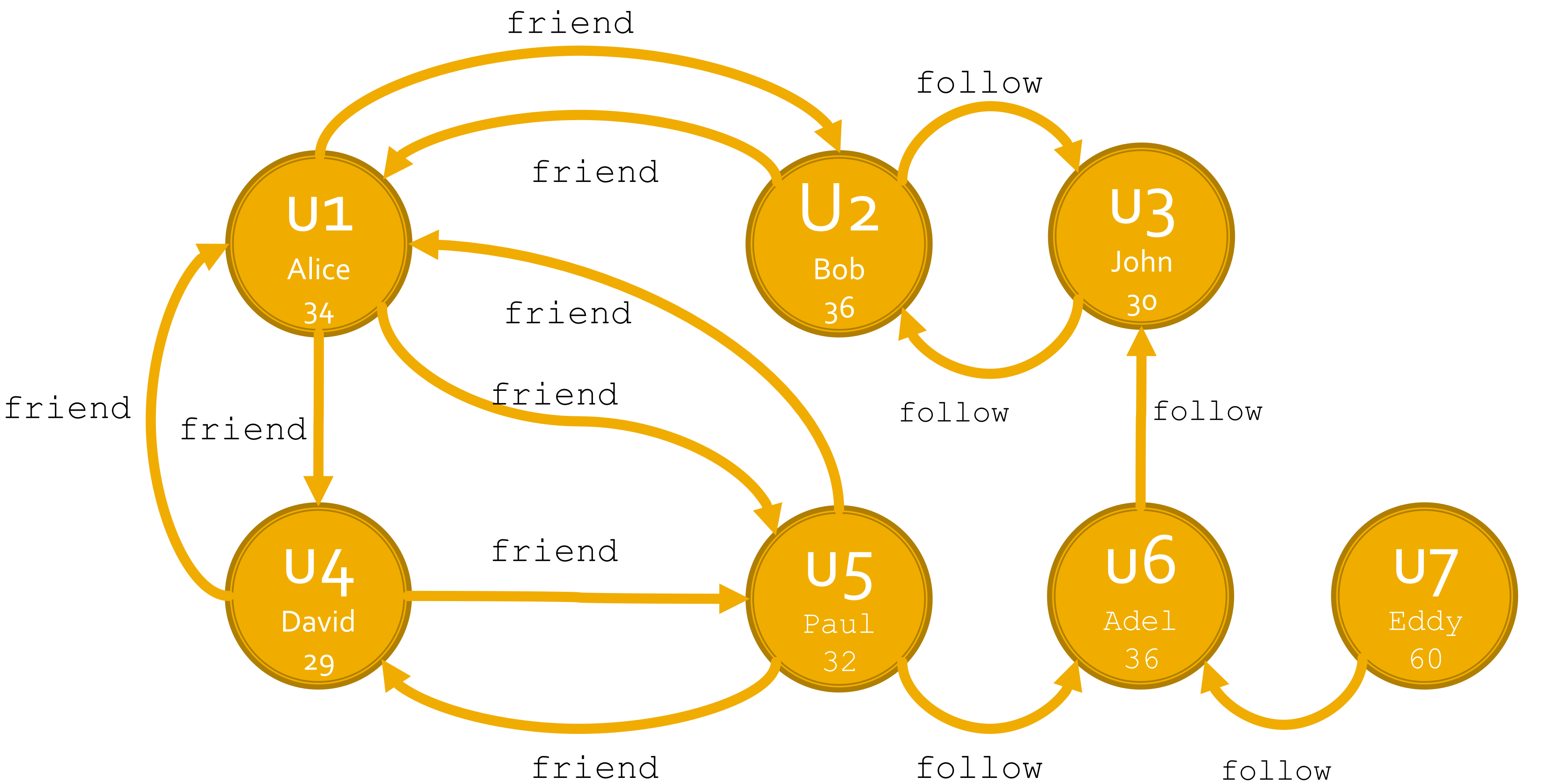
- The textual file edges.csv
  - It contains the edges of a graph
- Each edge is characterized by
  - src (string): source vertex
  - dst (string): destination vertex
  - linktype (string): "follow"or "friend"

Exercise #53

- Output:
  - Consider only the users with at least one follower
  - Store in the output folder the user(s) with the maximum number of followers
    - One user per line
    - Format: user id, number of followers
  - Use the CSV format to store the result

Exercise #53

Input graph example



Exercise #53

- Result

id	numFollowers
u3	2
u6	2

Exercise #54

- GraphFrame
- Input:
  - The textual file vertexes.csv
    - It contains the vertexes of a graph
  - Each vertex is characterized by
    - id (string): user identifier
    - name (string): user name
    - age (integer): user age

Exercise #54

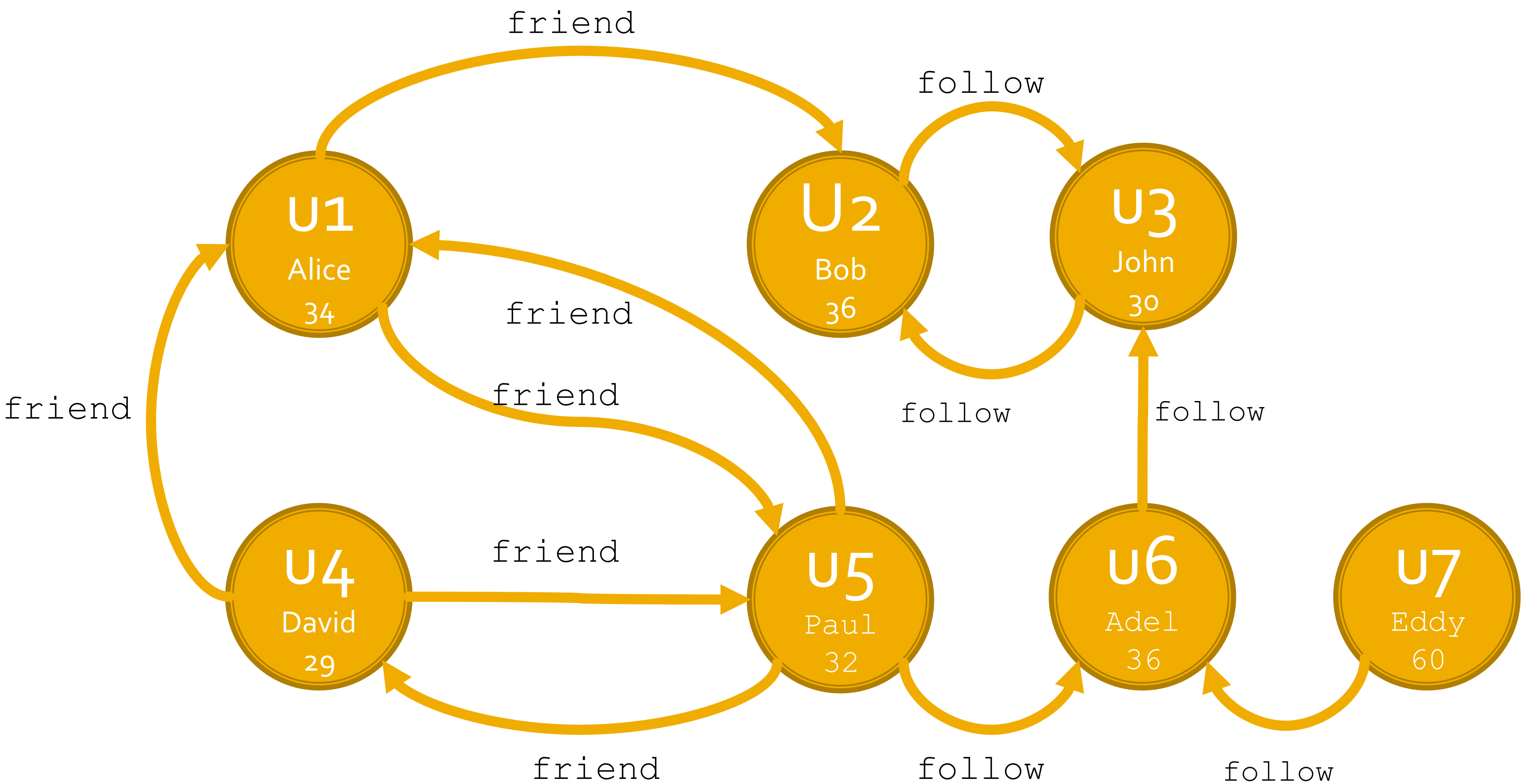
- The textual file edges.csv
  - It contains the edges of a graph
- Each edge is characterized by
  - src (string): source vertex
  - dst (string): destination vertex
  - linktype (string): “follow”or “friend”

Exercise #54

- Output:
  - The pairs of users Ux, Uy such that
    - Ux is friend of Uy (link “friend” from Ux to Uy)
    - Uy is not friend of Uy (no link “friend” from Uy to Ux)
  - One pair Ux,Uy per line
  - Format: idUx, idUy
  - Use the CSV format to store the result

Exercise #54

Input graph example



Exercise #54

- Result

IdFriend	IdNotFriend
U4	U1
U1	U2

Exercise #55

- GraphFrame
- Input:
  - The textual file vertexes.csv
    - It contains the vertexes of a graph
  - Each vertex is characterized by
    - id (string): vertex identifier
    - entityType (string): “user” or “topic”
    - name (string): name of the entity

Exercise #55

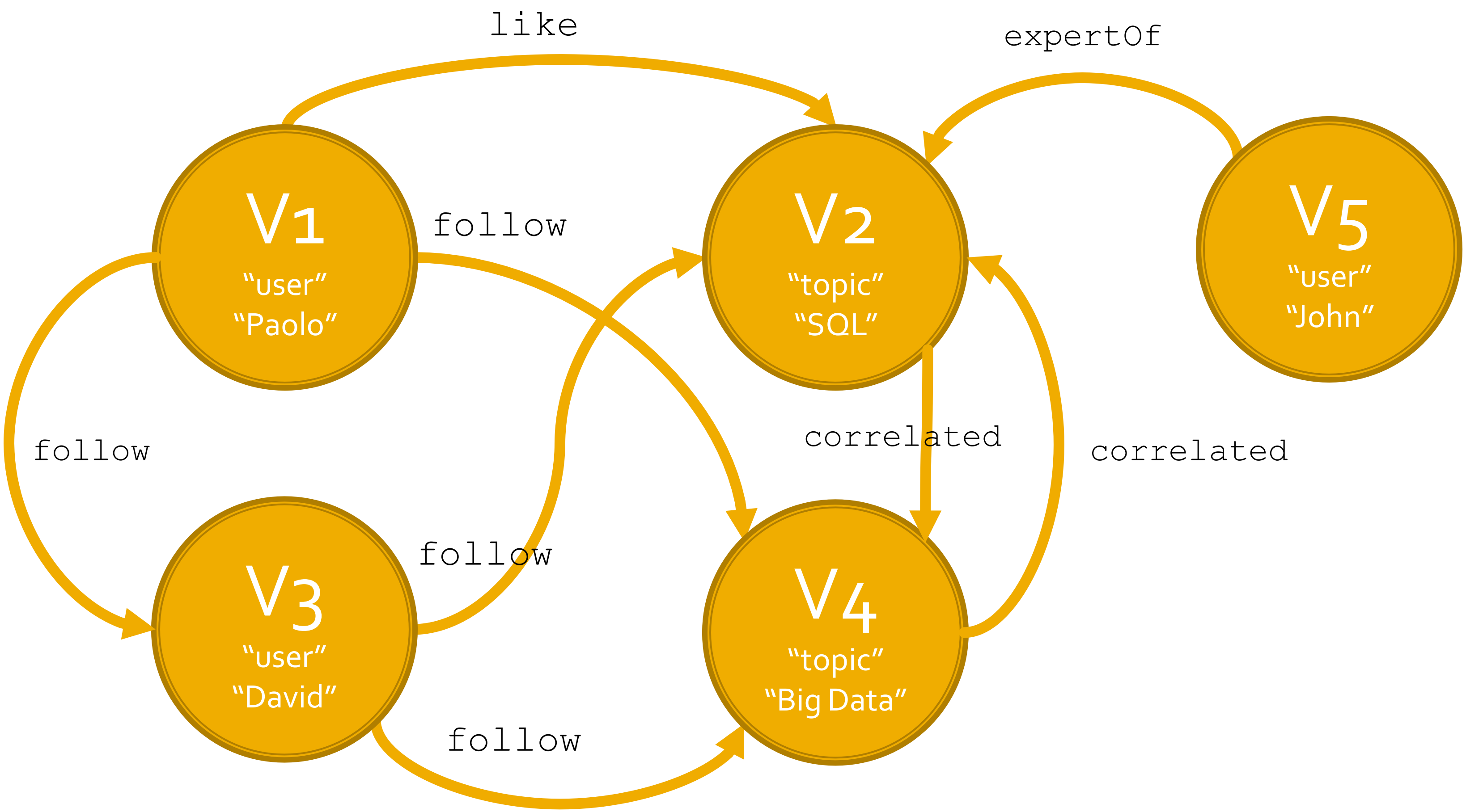
- The textual file edges.csv
  - It contains the edges of a graph
- Each edge is characterized by
  - src (string): source vertex
  - dst (string): destination vertex
  - linktype (string): “expertOf” or “follow” or “correlated”

Exercise #55

- Output:
  - The followed topics for each user
  - One pair (user name, followed topic) per line
  - Format: username, followed topic
  - Use the CSV format to store the result

Exercise #55

Input graph example



Exercise #55

- Result

username	topic
Paolo	Big Data
David	SQL
David	Big Data

Exercise #56

- GraphFrame
- Input:
  - The textual file vertexes.csv
    - It contains the vertexes of a graph
  - Each vertex is characterized by
    - id (string): vertex identifier
    - entityType (string): "user" or "topic"
    - name (string): name of the entity

Exercise #56

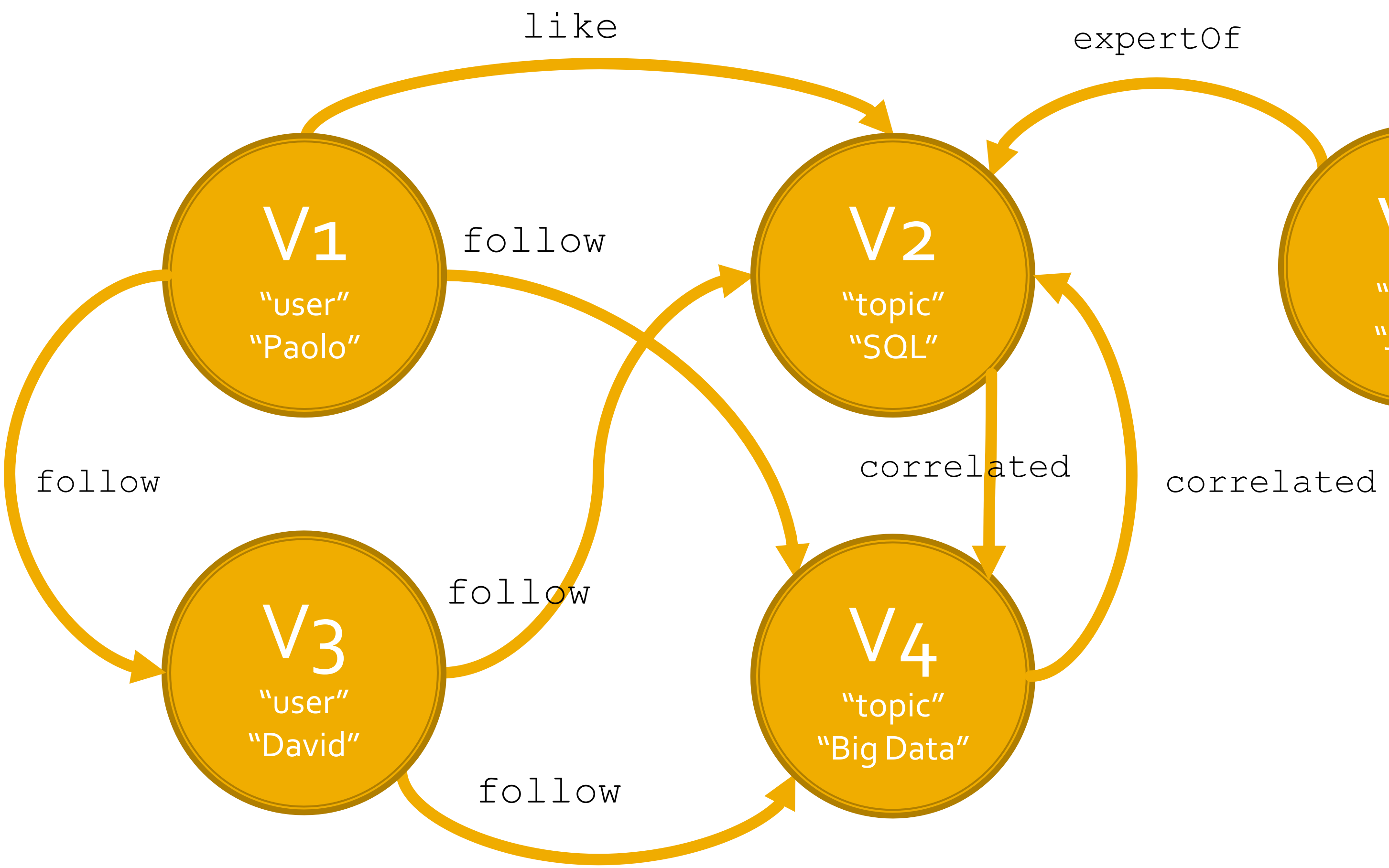
- The textual file edges.csv
  - It contains the edges of a graph
- Each edge is characterized by
  - src (string): source vertex
  - dst (string): destination vertex
  - linktype (string): "expertOf" or "follow" or "correlated"

Exercise #56

- Output:
  - The names of the users who follow a topic correlated with the “Big Data” topic
  - One user name per line
  - Format: username
  - Use the CSV format to store the result

Exercise #56

Input graph example



Exercise #56

- Result

username
David

Exercise #57

- GraphFrame
- Input:
  - The textual file vertexes.csv
    - It contains the vertexes of a graph
  - Each vertex is characterized by
    - id (string): user identifier
    - name (string): user name
    - age (integer): user age

Exercise #57

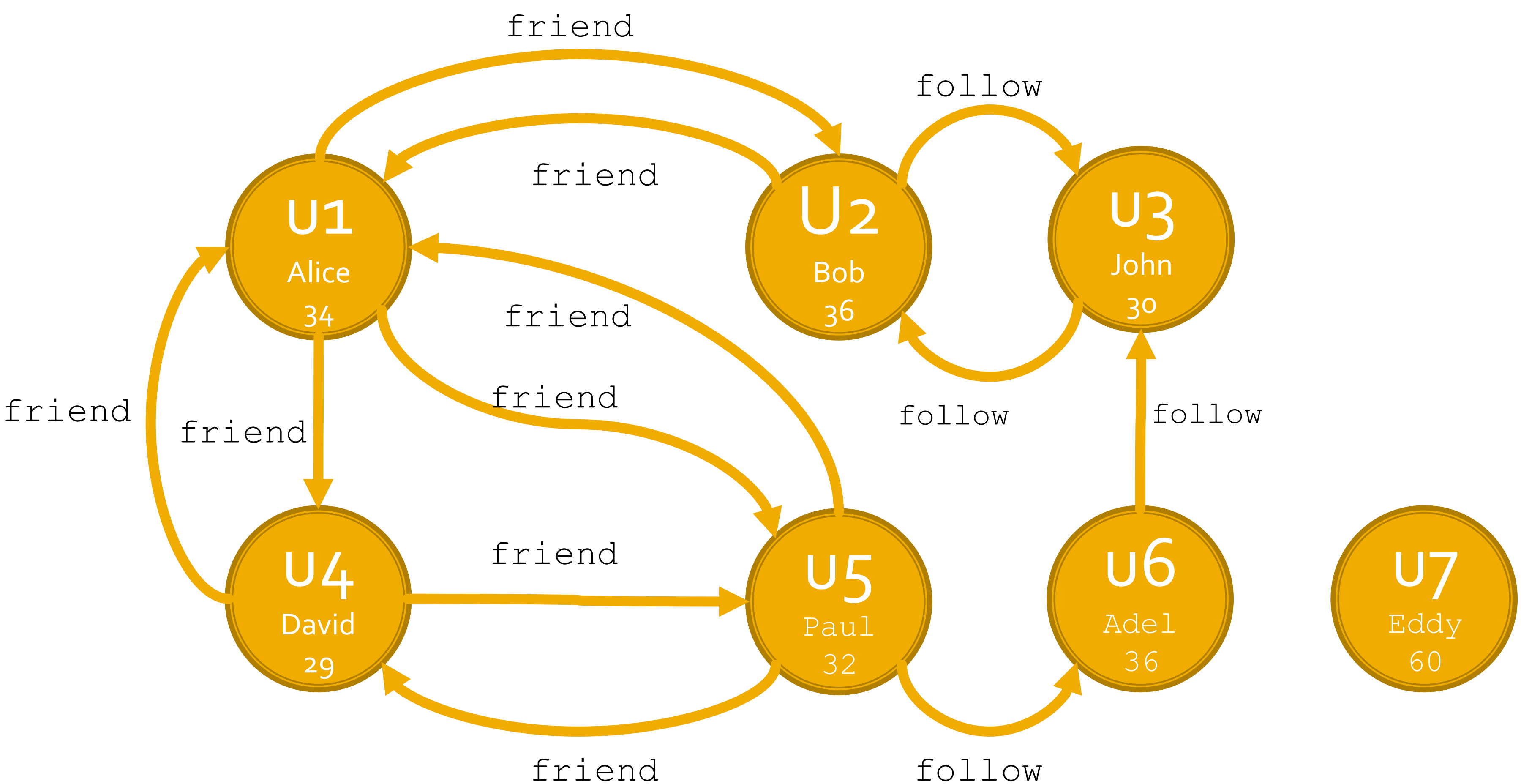
- The textual file edges.csv
  - It contains the edges of a graph
- Each edge is characterized by
  - src (string): source vertex
  - dst (string): destination vertex
  - linktype (string): “follow”or “friend”

Exercise #57

- Output:
  - Select the users who can reach user u1 in less than 3 hops (i.e., at most two edges)
    - Do not consider u1 itself
  - For each of the selected users, store in the output folder his/her name and the minimum number of hops to reach user u1
    - One user per line
    - Format: user name, #hops to user u1
  - Use the CSV format to store the result

Exercise #57

Input graph example



Exercise #57

- Result

name	numHops
Bob	1
John	2
David	1
Paul	1

Exercise #58

- Full station identification in real-time
- Input:
  - A stream of readings about the status of the stations of a bike sharing system
    - Each reading has the format
      - stationId,# free slots,#used slots,timestamp

Exercise #58

- Output:
  - For each reading with a number of free slots equal to 0
    - print on the standard output timestamp and stationId
  - Emit new results every 2 seconds by considering only the data received in the last 2 seconds

Exercise #59

- Full situation count in real-time
- Input:
  - A stream of readings about the status of the stations of a bike sharing system
    - Each reading has the format
      - stationId,# free slots,#used slots,timestamp

Exercise #59

- Output:
  - For each batch, print on the standard output the number of readings with a number of free slots equal to 0
  - Emit new results every 2 seconds by considering only the data received in the last 2 seconds

Exercise #60

- Full distinct stations identification in real-time
- Input:
  - A stream of readings about the status of the stations of a bike sharing system
    - Each reading has the format
      - stationId,# free slots,#used slots,timestamp

Exercise #60

- Output:
  - For each batch, print on the standard output the **distinct stationIds** associated with a reading with a number of free slots equal to 0 in each batch
  - Emit new results every 2 seconds by considering only the data received in the last 2 seconds

Exercise #61

- Maximum number of free slots in real-time
- Input:
  - A stream of readings about the status of the stations of a bike sharing system
    - Each reading has the format
      - stationId,# free slots,#used slots,timestamp

Exercise #61

- Output:
  - For each batch, print on the standard output the maximum value of the field "# free slots" by considering all the readings of the batch (independently of the stationId)
  - Emit new results every 2 seconds by considering only the data received in the last 2 seconds

Exercise #62

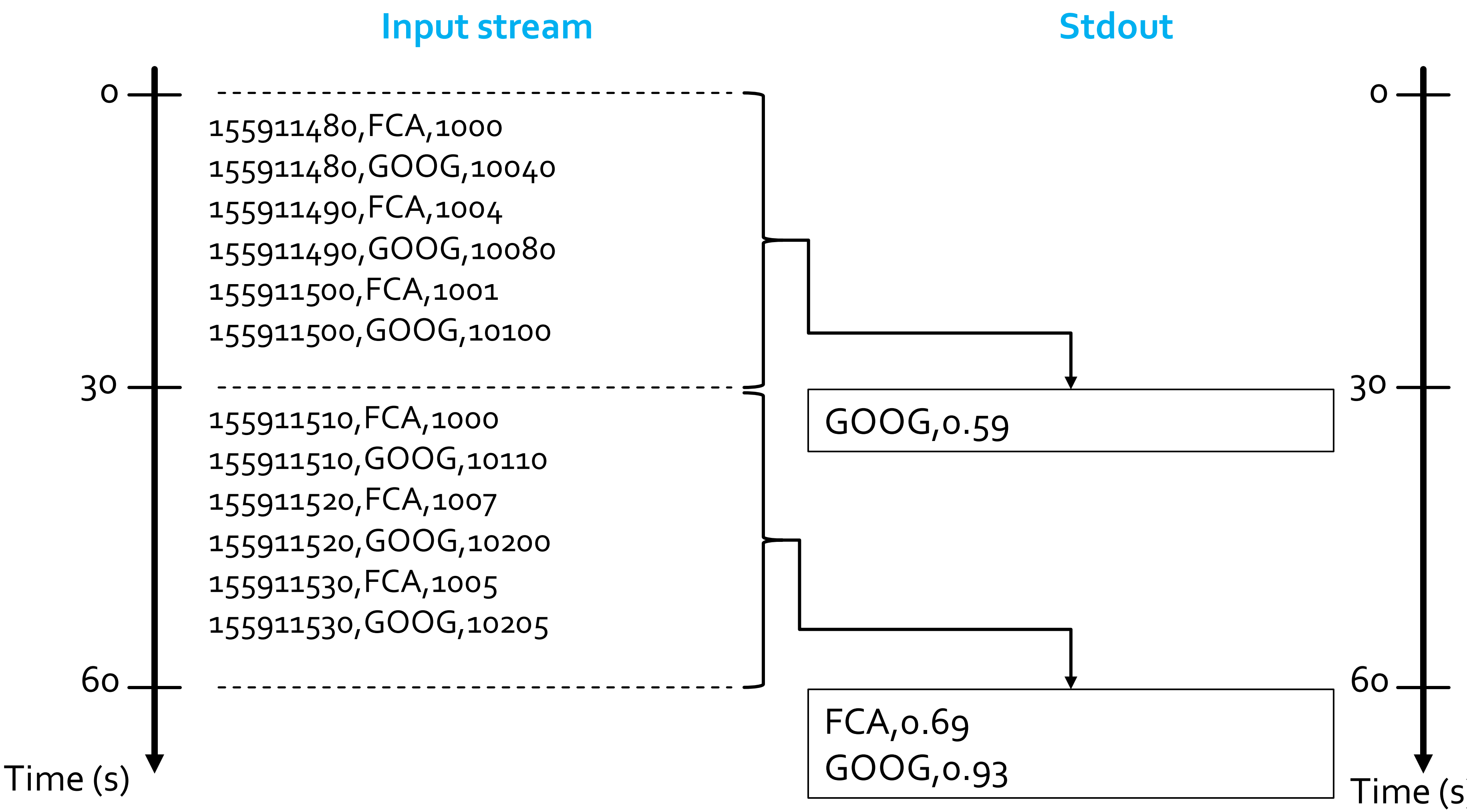
- High stock price variation identification in real-time
- Input:
  - A stream of stock prices
    - Each input record has the format
      - Timestamp,StockID,Price

Exercise #62

- Output:
  - Every 30 seconds print on the standard output the **StockID and the price variation (%) in the last 30 seconds** of the stocks with a **price variation greater than 0.5%** in the last 30 seconds
    - Given a stock, its price variation during the last 30 seconds is:

$$\frac{\max(price)-\min(price)}{\max(price)}$$

Exercise #62- Example



Exercise #62 Bis

- High stock price variation identification in real-time
- Input:
  - A stream of stock prices
    - Each input record has the format
      - Timestamp,StockID,Price

Exercise #62 Bis

- Output:
  - **Every 30 seconds** print on the standard output the **StockID and the price variation (%)** **in the last 60 seconds** of the stocks with a **price variation greater than 0.5%** **in the last 60 seconds**
    - Given a stock, its price variation during the last 60 seconds is:

$$\frac{\max(\text{price})-\min(\text{price})}{\max(\text{price})}$$

Exercise #63

- Full station identification in real-time
- Input:
  - A textual file containing the list of stations of a bike sharing system
    - Each line of the file contains the information about one station

      id**\t**longitude**\t**latitude**\t**name
  - A stream of readings about the status of the stations
    - Each reading has the format
      - StationId,# free slots,#used slots,timestamp

Exercise #63

- Output:
  - For each reading with a number of free slots equal to 0
    - print on the standard output timestamp and name of the station
  - Emit new results every 2 seconds by considering only the data received in the last 2 seconds

Exercise #64

- Anomalous stock price identification in real-time
- Input:
  - A textual file containing the historical information about stock prices in the last year
    - Each input record has the format
      - Timestamp,StockID,Price
  - A real time stream of stock prices
    - Each input record has the format
      - Timestamp,StockID,Price

Exercise #64

- Output:
  - Every 1 minute, by considering only the data received in the last 1 minute, print on the standard output the StockIDs of the stocks that satisfy one of the following conditions
    - price of the stock (received on the real-time input data stream) < historical minimum price of that stock (based only on the historical file)
    - price of the stock (received on the real-time input data stream) > historical maximum price of that stock (based only on the historical file)
  - If a stock satisfies the conditions multiple times in the same batch, return the stockId only one time for each batch

Exercise #64- Example

- Textual file containing the historical information about stock prices in the last year

  130000000,FCA,1000

  130000000,GOOG,10040

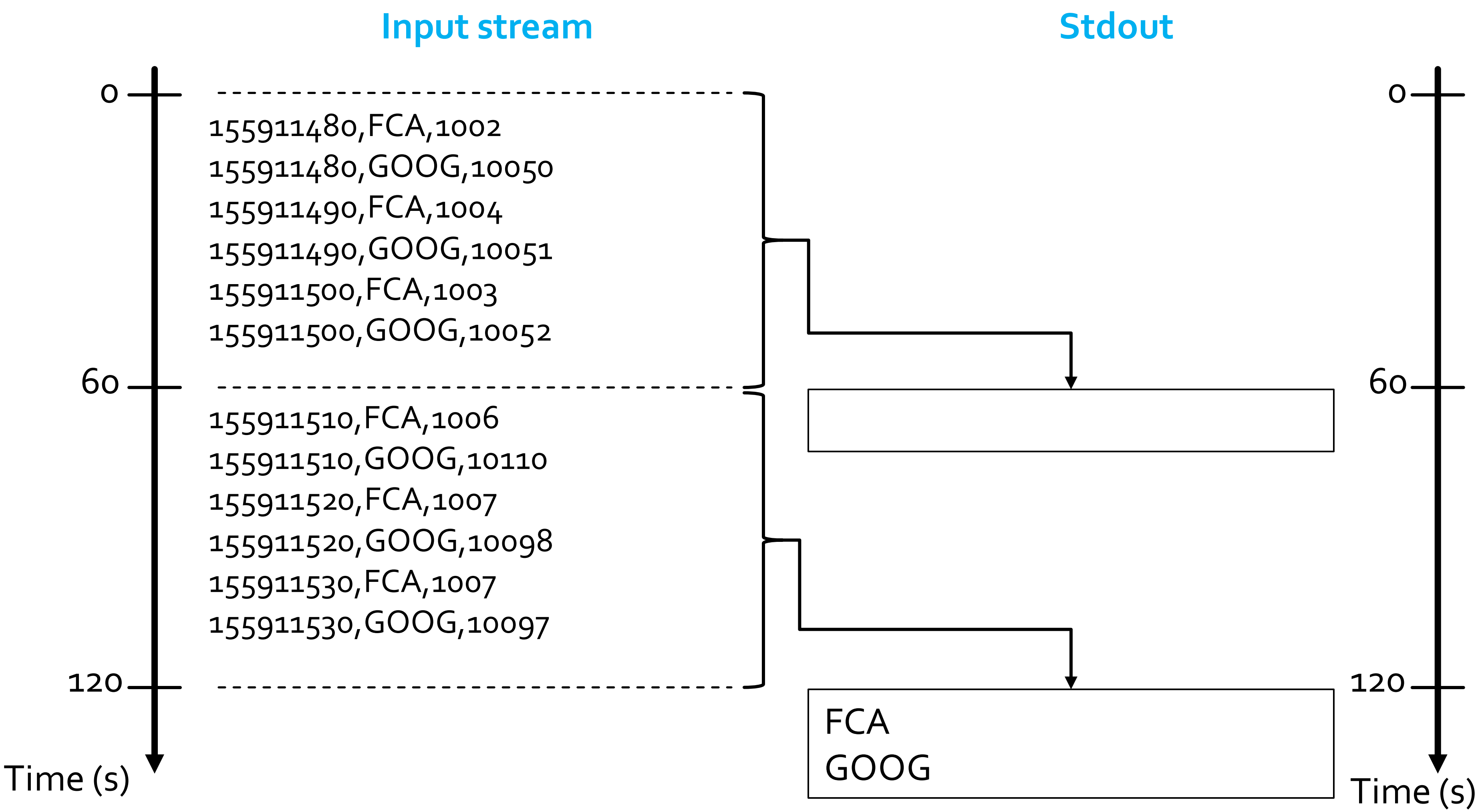
  130000060,FCA,1004

  130000060,GOOG,10080

  130000120,FCA,1001

  130000120,GOOG,10100

Exercise #64- Example



Exercise #65

- Anomalous stock price identification in real-time
- Input:
  - A textual file containing the historical information about stock prices in the last year
    - Each input record has the format
      - Timestamp,StockID,Price
  - A real time stream of stock prices
    - Each input record has the format
      - Timestamp,StockID,Price

Exercise #65

- Output:
  - Every 30 seconds, by considering only the data received in the last 1 minute, print on the standard output the StockIDs of the stocks that satisfy one of the following conditions
    - price of the stock (received on the real-time input data stream) < historical minimum price of that stock (based only on the historical file)
    - price of the stock (received on the real-time input data stream) > historical maximum price of that stock (based only on the historical file)
  - If a stock satisfies the conditions multiple times in the same batch, return the stockId only one time for each batch

Exercise #65- Example

- Textual file containing the historical information about stock prices in the last year

  130000000,FCA,1000

  130000000,GOOG,10040

  130000060,FCA,1004

  130000060,GOOG,10080

  130000120,FCA,1001

  130000120,GOOG,10100

Exercise #65- Example