YOUNICAM for COVID Analysis using Spark GraphX

Federico Fabrizi, 115928

UNICAM, MD in Computer Science, Data Analytics 20-21

Goals

The objective of this project is to perform **data analysis** on the data collected with the YOUNICAM App.

Analysis must be related to an ipotetic scenario in which a student registered by the App is tested positive for COVID.

The analysis should be able to identify the cluster of people connected to the positive student and the location where the contacts happen.

Analysis must be implemented by using Spark GraphX as a batch analytics job.

Spark GraphX

GraphX is a Spark component for graphs and graph-parallel computation.

At a high level, GraphX extends the Spark RDD by introducing a new Graph abstraction: a **directed multigraph** with **properties** attached to each vertex and edge. To support graph computation, GraphX exposes a set of **fundamental operators**(e.g., subgraph, joinVertices, and aggregateMessages).

In addition, GraphX includes a growing collection of **graph algorithms** and **builders** to simplify graph analytics tasks.

Property Graphs

The **property graph** is a directed multigraph with **user defined objects** attached to each vertex and edge. A directed multigraph is a directed graph with potentially multiple parallel edges sharing the same source and destination vertex.

The ability to support parallel edges simplifies modeling scenarios where there can be **multiple relationships** (e.g., co-worker and friend) between the same vertices. Each vertex is keyed by a unique 64-bit long identifier (VertexId). GraphX does not impose any ordering constraints on the vertex identifiers. Similarly, edges have corresponding source and destination vertex identifiers.

Datasets

The **datasets** used in this project represent data about

- single **presences** of students: student, datetime, classroom, seat...
- **classrooms**: id, location, number of seats...

_id	id	username	aula	sede	polo	inDate	date
5fa8ef7d1bd2a03f4641a15e	81929	1	1	1	1	11/9/20 8:27	11/9/20
5fa8efa51bd2a03f4641a15f	81098	2	1	1	1	11/9/20 8:28	11/9/20
5fa8f0751bd2a03f4641a160	81009	3	1	1	1	11/9/20 8:32	11/9/20
5fa8f0811bd2a03f4641a161	80492	4	1	1	1	11/9/20 8:32	11/9/20
5fa8f0891bd2a03f4641a162	80492	4	1	1	1	11/9/20 8:32	11/9/20
5fa8f16b1bd2a03f4641a163	81974	5	1	1	1	11/9/20 8:36	11/9/20
5fa8f17d1bd2a03f4641a164	80436	6	2	1	2	11/9/20 8:36	11/9/20
5fa8f1951bd2a03f4641a165	81613	7	2	1	2	11/9/20 8:36	11/9/20

Sede	_id	aulaDes	aulaId	capienza
1	5f5b4fdd400b9fbfbe0b8e94	69	1	10
1	5f5b4fdd400b9fbfbe0b8e95	25	2	24
1	5f5b4fdd400b9fbfbe0b8e96	12	3	12
1	5f5b4fdd400b9fbfbe0b8e97	16	4	10
1	5f5b4fdd400b9fbfbe0b8e98	35	7	76
1	5f5b4fdd400b9fbfbe0b8e99	1	1	75
1	5f5b4fdd400b9fbfbe0b8e9a	19	2	12
1	5f5b4fdd400b9fbfbe0b8e9b	14	3	12

Logic

Exploiting **simple graph modeling** for computing which students have seated next to a student that tested positive to SARS-CoV-2 infection and should thus be warned, under **parametrized risk criteria**. This is done in three steps:

1. **Preparing** the data
2. **Building** the Graph
3. Graph **Analysis**

Risk Criteria

“Immuni” is a mobile app that allows users to be notified if they come in contact with someone that has been tested positive to SARS-CoV-2 and decided to insert this information into the system. (https://www.immuni.italia.it/)

The system guarantees that its users can remain anonymous and uses a bluetooth-based calculation to determine the distance between two smartphones.

Its **risk criteria** suggest that remaining **within 2 meters for more than 15 minutes** near a positive person should be considered worth of an alert.
(https://www.ilsole24ore.com/art/immuni-come-funziona-l-algoritmo-che-misura-rischio--ADeCP9W)

Project: Functionality

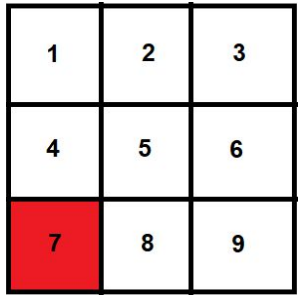
This project is realized in terms of two Scala classes, that have a main method and act as a script:

- The class **GraphBuilder** uses the “presences.csv” dataset to generate a **property graph** of the presences, on which analysis then is executed
- The class **GraphAnalyzer** uses the output of the GraphBuilder class to filter the graph and determine if some students should be warned
    - **search** is based on students’ **id** and on their **seat number** in the classroom
    - **risk criteria** are function parameters
    - distance between students is determined based on the **adjacent seats** to a given one

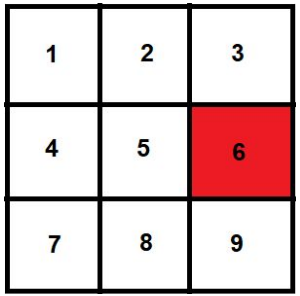
Adjacent Seats

Currently, seat numbers are assigned sequentially, based on the **total number of seats** of a classroom and a **row length parameter**, that is how many seats per row a classroom has.
Adjacent seats are all considered within a risky range. These **assumptions** are just for the sake of **simplicity**: in a real implementation, these informations should be retrieved from the dataset.

Given a **seat number**, a function computes all its **neighbor seats**(max 8), for example:



Neighbors:
[4, 5, 8]



Neighbors:
[2, 3, 5, 8, 9]

1. Preparing the Data

Presences data is loaded into a Spark **dataframe**, that is an higher level data structures respect to RDDs, that supports SQL-like operations.

Only relevant fields are retained: presence_id, student_id, seat number, classroom id, Polo, Sede, and in and out timestamps.

Timestamps are parsed into a datetime format, so that they can
- be compared to check if they refer to the same day
- be converted into seconds from Epoch time to execute computations on time intervals

(https://spark.apache.org/docs/3.0.1/api/sql/#unix_timestamp)

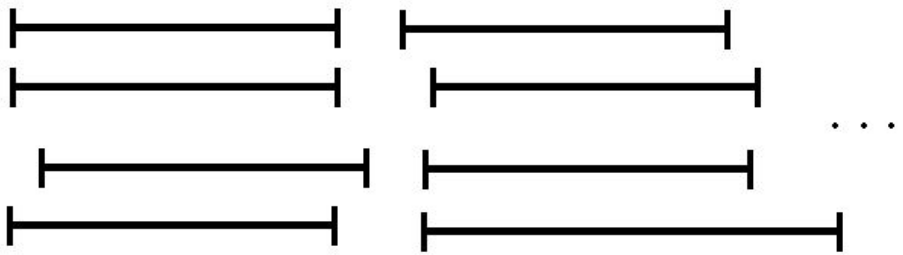
2. Building the Graph

The property graph if built from **RDDs**, directly transforming the **dataframes** and defining **vertices** and **edges** separately, then using a Graph object.
This is just one of the ways of defining a graph that GraphX supports.

- **Vertices** will contain all the information related to a **single presence**
- **Edges** will connect two Vertices if they refer to the **same classroom** in the **same day**
- Edges will have a **property** that stands for the number of **minutes** in which the two linked presences overlap
    - a function considers all the possible ways two time intervals can overlap and returns an integer that stands for and amount in minutes

Overlapping Time Intervals

A function considers all the possible ways two time intervals taken in input can overlap and returns an integer that stands for an amount of minutes. Specifically, **intervals** are specified in terms of **start** and **end** timestamps, converted in seconds from Epoch time.



3. Graph Analysis

In this phase the **property graph** that contains all the necessary information is analyzed, based on **risk criteria** and the **student id** of the student that tested positive, taken in input.

The **analysis** consists in examining the **neighbor vertices** of every vertex that stands for a presence of the student that tested positive: those vertices refer to other presences in the same classroom in the same day. If those also fall within **risk criteria**, the students they refer to should receive an alert.

GraphX allows an efficient use of the graph structure, providing operations for **filtering** vertices, creating **subgraphs**, executing **map operations** on vertices values, **transforming** vertices and so on.

Project: Implementation

All the computation is realized inside **SparkSessions** using only methods that preserve distributed computation, and the execution can be potentially assigned to a **cluster** by providing a **master parameter**.

Some **assumptions** that are made, regarding the disposition of the seats and missing data from the datasets, should be further evaluated in a real implementation, expanding the functionalities of the current version.

Project: Technologies

The project is realized as a **Scala sbt project** using **IntelliJ** and **Spark** in a standalone installation. Precisely:

- Scala 2.12.12
- Sbt 1.4.7
- Spark 3.0.1
- Java 15 JDK

These **versions** are relevant since there could be **incompatibility issues** among them, resulting in the impossibility of building and running the code.
If you already have a different version of any of them, make sure to check compatibility.

Project: Technologies

Prerequisites for running the project:

- Having an **IntelliJ** installation with the **Scala** plugin
- Having a **Java SDK** installation
- Having a **Spark standalone** installation

In case of incompatibility issues, or if you need to use different versions of these technologies, refer to https://mvnrepository.com/ to configure the dependencies inside the sbt configuration accordingly.

Running the project

The **sbt project** is available at
https://github.com/FedericoFabriziIT/YOUNICAM-COVID-Spark-Analysis
and can be ran directly inside the IntelliJ IDE, simply by downloading it and opening it, after waiting for it to compile and retrieve the necessary modules.
The directory “data” already contains the sample **datasets**.

- It may need changes in the **project structure properties**, to set the correct JDK and Scala versions
- The first time will take some time to build the project from the **sbt build file**

Running the project

Execution steps:

1. **Invoke** the execution of “GraphBuilder.scala”
2. **Invoke** the execution of “GraphAnalyzer.scala”
3. **Check** the results in the “ExposedStudents.txt” file

Running the project

1. Invoke the execution of “GraphBuilder.scala”:

- it will prompt for a **master parameter**, default to “local[*]”
- it will prompt for the **presences file path**, default to “data/presences.csv”
- it will execute the **data preparing step** and build a **graph** that will be persisted by creating the directories “vertices” and “edges”
    - those directories must not exist before the execution

Running the project

2. Invoke the execution of “GraphAnalyzer.scala”:

- it will prompt for a **master parameter**, default to “local[*]”
- it will prompt for the **rooms file path**, default to “data/rooms.json”
- it will prompt for the **parameters** student_id, minutes and days
  - they have to be provided separated by a whitespace
  - make sure to provide an high enough value for the “days” parameter, since the presences dataset contains data up to December 2020 at most
- it will create the file “ExposedStudents.txt”, containing the result of the analysis

Parameters:
- **student_id**: the id of the student that tested positive
- **minutes**: determines over how many minutes a presence near a positive student should be considered at risk
- **days**: determines within how many days in the past presences should be considered for the analysis

Running the project

3. Check the results in the “ExposedStudents.txt” file:

- the file output file contains in every line information about
  - the **student** that should be warned
  - the **location** and the **date** of the presence that was considered at risk
  - the two **ids** referred to the two **presences** that were two adjacent vertices in the graph

```
1  These students may have been exposed to SARS-COV-2 infection and should be warned:
2  (Every line refers to a single student and includes where the contact has happened.)
3  Studente: 80560, Aula: 3, Polo: 3, Sede: 1, Data: 12/21/20 9:03 - Presences' Ids: 5fe056bb1c7ca18763114bd1, 5fe04fee1c
4  Studente: 81851, Aula: 3, Polo: 3, Sede: 1, Data: 12/2/20 9:06 - Presences' Ids: 5fc74b1421015899996e8ff8, 5fc749eb210
5  Studente: 80871, Aula: 3, Polo: 3, Sede: 1, Data: 12/2/20 9:06 - Presences' Ids: 5fc74b1421015899996e8ff8, 5fc74ba3210
6  Studente: 558, Aula: 3, Polo: 3, Sede: 1, Data: 12/21/20 9:03 - Presences' Ids: 5fe056bb1c7ca18763114bd1, 5fe056351c7c
```

References

- Spark GraphX: https://spark.apache.org/docs/3.0.1/index.html
- Intellij IDE: https://www.jetbrains.com/idea/
- Java SDK: https://www.oracle.com/java/technologies/javase/jdk15-archive-downloads.html
- SBT: https://www.scala-sbt.org/