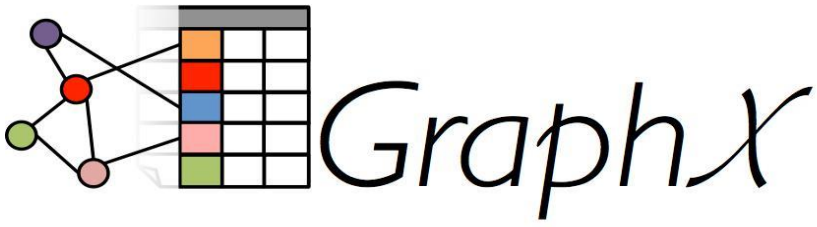
ANALYSE DES PROFILS LINKEDIN AVEC GRAPHX

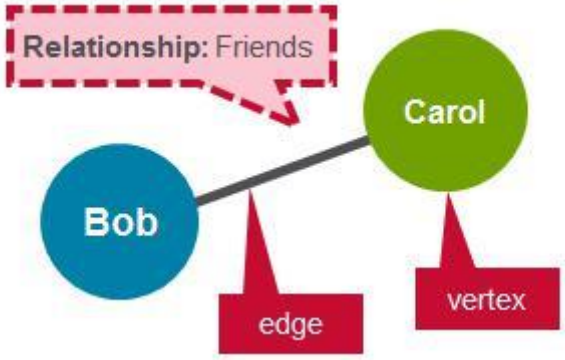


SPARK GRAPHX :

Avec l'émergence d'Apache Spark, l'analyse des méga données est devenue plus facile, Spark apporte beaucoup d'implémentation d'algorithmes utiles pour l'exploration de données, l'analyse de données, l'apprentissage automatique, les algorithmes sur les graphiques.

Spark - GraphX, c'est un composant pour les graphiques et le calcul parallèle au graphique. GraphX réutilise le concept Spark RDD, simplifie les tâches d'analyse graphique, offre la possibilité d'effectuer des opérations sur une multi graphe dirigée avec des propriétés attachées à chaque sommet (vertex) et arête (edge). GraphX fournit une API pour un développement rapide et robuste lié à l'exploitation des graphiques.

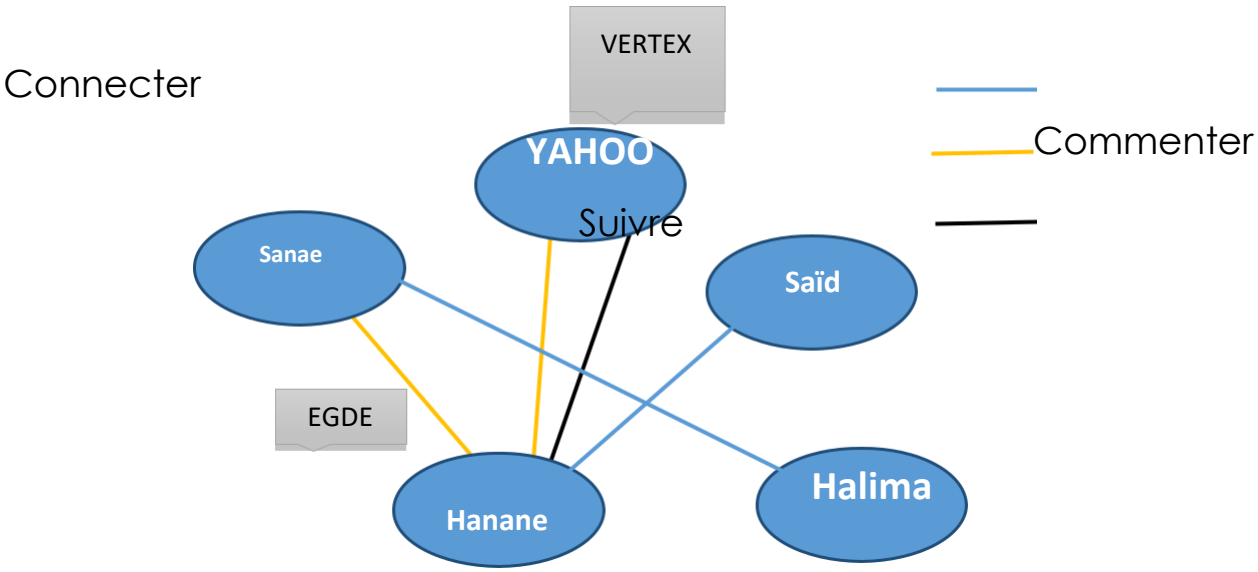
Pour chaque graphe, on a deux composantes« vertex » et « edge » exprimés comme suit :



C'est comme notre cas, on a les vertex sont les personnes et les entreprises dans LinkedIn et les edges sont simplement la relation connectée.

Notre Exemple :

On prend 5 personnes sur LinkedIn interconnectes entre eux :



Donc, on va avoir deux tableaux le premier ce qui définit la liste des vertex et le deuxième c'est des edges :

ID	NOM
1	yahoo
2	sanae
3	hanane
4	said
5	halima

Vertex 1	Vertex 2	relation
1	3	commenter
1	3	suivre
2	3	commenter
2	5	connecter
4	2	connecer

PS : Pour les relations suivre et commenter on va dire qu'ils sont de deux côtes.

En scala on définit les vertex par le code suivant :

```
importorg.apache.spark._
importorg.apache.spark.rdd.RDD
// import classes required for using
GraphX importorg.apache.spark.graphx._
/ create vertices RDD with ID and Name
val vertices=Array((1, ("yahoo")),(2,
("sanae")),(3,("hanane")),(4,("said")),(5,("halima")))
valvRDD= sc.parallelize(vertices)
vRDD.take(1)
// Defining a default vertex called
nowhere valnowhere = "nowhere"
```

Pour les edges on a le code suivant :

```
// create routes RDD with srcid, destid, distance
val edges =
Array(Edge(1,3,'commenter'),Edge(1,5,'suivre'),Edge(2,3,'commente
r'),Edge(2,5,'connecter'),Edge(4,2,'connecter'))
valeRDD= sc.parallelize(edges)
eRDD.take(2)
```

pourcréer le graphe on a le code suivant:

```
// define the graph
val graph = Graph(vRDD,eRDD, nowhere)
// graph vertices
graph.vertices.collect.foreach(println)
```

```
// (1,'yahoo')
//…
// graph edges
graph.edges.collect.foreach(println)
// Edge(1,3,'commenter')
// …
```

Après la définition du graphe on peut avoir des réponses sur plusieurs questions.

Combien de compte (personne et entreprise) sur LinkedIn?

```
valnumaccounts =
graph.numVertices // Long = 5
```

Combien de routes sur LinkedIn?

```
valnumroutes = graph.numEdges
// Long = 5
```

Implémentation d'un exemple réel en Sparkshell:

A cause de l'indisponibilité d'une data set de la part du LinkedIn et parce que nous avons fait l'exemple précèdent qui va être similaire pour le cas de LinkedIn mais simplement avec une data set large de donnée, nous avons changé le sujet de l'analyse vers une analyse d'une data set de vol.

- ✓ Scenario :

Nous avons tiré le data set depuis le lien suivant :
https://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_

Short_Name=On-Time Pour chaque vol nous avons les informations suivantes:

CHAMP	DESCRIPTION	VALEUR
dOfM(String)	Day of month	1
dOfW (String)	Day of week	4
carrier (String)	Carrier code	AA
tailNum (String)	Unique identifier for the plane - tail number	N787AA
flnum(Int)	Flight number	21
org_id(String)	Originairport ID	12478
origin(String)	Origin Airport Code	JFK
dest_id (String)	Destination airport ID	12892

dest (String)	Destination airport code	LAX
crsdeptime(Double)	Scheduleddeparture time	900
deptime (Double)	Actualdeparture time	855
depdelaymins (Double)	Departuredelay in minutes	0
crsarrtime (Double)	Scheduledarrival time	1230
arrtime (Double)	Actualarrival time	1237
arrdelaymins (Double)	Arrivaldelay minutes	7
crselapsedtime (Double)	Elapsed time	390
dist (Int)	Distance	2475

Dans ce scénario, nous allons représenter les aéroports comme des vertices et des itinéraires comme des bords. Nous sommes intéressés à visualiser les aéroports et les itinéraires et nous aimerions voir le nombre d'aéroports qui ont des départs ou des arrivées.

- ✓ Le code:

Démarrez le Spark Shell avec :

```
$ Spark-shell
```

- Définition des Vertex :

D'abord, nous importerons les paquets GraphX.

```
importorg.apache.spark._
importorg.apache.spark.rdd.RDD
importorg.apache.spark.util.IntParam
importorg.apache.spark.graphx._
importorg.apache.spark.graphx.util.GraphGenerators
```

Ci-dessous nous utilisons les classes de cas Scala pour définir le schéma de vol correspondant au fichier de données csv.

```
case class Flight(dofM:String, dofW:String, carrier:String, tailnum:String,
flnum:Int, org_id:Long, origin:String, dest_id:Long, dest:String,
crsdeptime:Double, deptime:Double, depdelaymins:Double,
crsarrtime:Double, arrtime:Double,
arrdelay:Double,crselapsedtime:Double,dist:Int)
```

La fonction ci-dessous analyse une ligne du fichier de données dans la classe de vol.

```
// function to parse input into Flight
class defparseFlight(str: String): Flight = {
val line = str.split(",")
 Flight(line(0), line(1), line(2), line(3), line(4).toInt, line(5).toLong,
line(6), line(7).toLong, line(8), line(9).toDouble, line(10).toDouble,
line(11).toDouble, line(12).toDouble, line(13).toDouble,
line(14).toDouble, line(15).toDouble, line(16).toInt)
}
```

Ci-dessous, nous chargeons les données du fichier csv dans un ensemble de données distribuées résilientes (RDD). Les RDD peuvent avoir des transformations et des actions, la première action renvoie le premier élément dans le RDD.

```
valtextRDD = sc.textFile("/user/user01/data/rita2014jan.csv")


valflightsRDD = textRDD.map(parseFlight).cache()
```

Nous définissons les aéroports comme des vertex. Les vertex peuvent avoir des propriétés ou des attributs qui leur sont associés. Chaque vertex a la propriété suivante: Nom de l'aéroport (String)

La table des vertes pour les aéroports :

ID	Property(V)
10397	ATL

Nous définissons un RDD avec les propriétés ci-dessus qui est ensuite utilisé pour les vertèbres.

```
// créer des aéroports RDD avec ID et Nom.
val airports = flightsRDD.map(flight =>
(flight.org_id, flight.origin)).distinct
airports.take(1)


// Définition d'un sommet par défaut appelé nulle
part val nowhere = "nowhere"


// Map airport ID to the 3-letter code to use for printlns
valairportMap = airports.map { case ((org_id), name) => (org_id -
> name) }.collect.toList.toMap
// Map(13024 -> LMT, 10785 -> BTV,…)
```

Définition desEdges

Les edges sont les routes entre les aéroports. Un bord doit avoir une source, une destination, et peut avoir des propriétés. Dans notre exemple, un edge se compose de :

Edgeorigin id → src (Long)

Edge destination id → dest (Long)

Edge property distance → distance (Long)

La table des edges pour les routes :

srcid	destid	Property(E)
14869	14683	1087

Nous définissons un RDD avec les propriétés ci-dessus qui est ensuite utilisé pour les bords. L'edgeRDD a la forme (src id, dest id, distance).

```
// create routes RDD with srcid, destid, distance
val routes = flightsRDD.map(flight => ((flight.org_id,
flight.dest_id), flight.dist)).distinctdistinct


routes.take(2)
// Array(((14869,14683),1087), ((14683,14771),1482))


// create edges RDD with srcid, destid ,
distance val edges = routes.map {
case ((org_id, dest_id), distance)
=>Edge(org_id.toLong, dest_id.toLong, distance) }


edges.take(1)
//Array(Edge(10299,10926,160))
```

Création du graph :.

Create a Pour créer un graphique, vous devez avoir un Vertex RDD, Edge RDD et un vertex par défaut.

```
// define the graph
val graph = Graph(airports, edges, nowhere)


// graph vertices
graph.vertices.take(2)
Array((10208,AGS), (10268,ALO))


// graph edges
graph.edges.take(2)
Array(Edge(10135,10397,692), Edge(10135,13930,654))
```

Phase finale (Les questions) :

Après qu'on a notre graphe prêt à être interrogé, voici une liste des exemples de questions qu'on peut poser :

Combien d'aéroports y a-t-il?

```
valnumairports =
graph.numVertices // Long = 301
```

Combien de routes y a-t-il?

```
valnumroutes = graph.numEdges
```

```
// Long = 4090
```

Quelles routes de distances>1000 miles?

```
graph.edges.filter{ case ( Edge(org_id, dest_id,distance))=>
distance > 1000}.take(3)
// Array(Edge(10140,10397,1269),
Edge(10140,10821,1670), Edge(10140,12264,1628))
```

La classe EdgeTriplet étend la classe de bord en ajoutant les membres srcAttr et dstAttr qui contiennent respectivement les propriétés source et destination.

```
// triplets
graph.triplets.take(3).foreach(println)
((10135,ABE),(10397,ATL),692)
((10135,ABE),(13930,ORD),654)
((10140,ABQ),(10397,ATL),1269)
```

Trier et imprimer les itinéraires les plus longs

```
// print out longest routes
graph.triplets.sortBy(_.attr, ascending=false).map(triplet =>
    "Distance " + triplet.attr.toString + " from " + triplet.srcAttr + " to "
+ triplet.dstAttr + ".").take(10).foreach(println)


Distance 4983 from JFK to HNL.
Distance 4983 from HNL to JFK.
Distance 4963 from EWR to HNL.
Distance 4963 from HNL to EWR.
```

```
Distance 4817 from HNL to IAD.
Distance 4817 from IAD to HNL.
Distance 4502 from ATL to HNL.
Distance 4502 from HNL to ATL.
Distance 4243 from HNL to ORD.
Distance 4243 from ORD to HNL.
```

Calculer le vertex le plus haut degré

```
// Define a reduce operation to compute the highest degree
vertex def max(a: (VertexId, Int), b: (VertexId, Int)): (VertexId, Int) = {
if (a._2 > b._2) a else b
}
valmaxInDegree: (VertexId, Int) = graph.inDegrees.reduce(max)
//maxInDegree: (org.apache.spark.graphx.VertexId, Int)
= (10397,152)

valmaxOutDegree: (VertexId, Int) = graph.outDegrees.reduce(max)
//maxOutDegree: (org.apache.spark.graphx.VertexId, Int)
= (10397,153)

valmaxDegrees: (VertexId, Int) = graph.degrees.reduce(max)
//maxDegrees: (org.apache.spark.graphx.VertexId, Int) = (10397,305)

// Get the name for the airport with id
10397 airportMap(10397)
//res70: String = ATL
```

Quel aéroport a le plus de vols entrants?

```
// get top 3
valmaxIncoming = graph.inDegrees.collect.sortWith(_._2
> _._2).map(x => (airportMap(x._1), x._2)).take(3)


maxIncoming.foreach(println)
(ATL,152)
(ORD,145)
(DFW,143)


**// which airport has the most outgoing flights?**
valmaxout= graph.outDegrees.join(airports).sortBy(_._2._1,
ascending=false).take(3)


maxout.foreach(println)
(10397,(153,ATL))
(13930,(146,ORD))
(11298,(143,DFW))
```

pour déterminer quels aéroports sont les plus importants en mesurant quels aéroports ont le plus de connexions avec d'autres aéroports. Nous devons préciser la tolérance, qui est la mesure de la convergence.

Quellssont les aéroports les plus importants selon PageRank?

```
// use pageRank
val ranks = graph.pageRank(0.1).vertices
// join the ranks with the map of airport id to name
val temp= ranks.join(airports)
temp.take(1)
// Array((15370,(0.5365013694244737,TUL)))


// sort by ranking
val temp2 = temp.sortBy(_._2._1, false)
temp2.take(2)
//Array((10397,(5.431032677813346,ATL)),
(13930,(5.4148119418905765,ORD)))


// get just the airport names
valimpAirports =temp2.map(_._2._2)
impAirports.take(4)
//res6: Array[String] = Array(ATL, ORD, DFW, DEN)
```