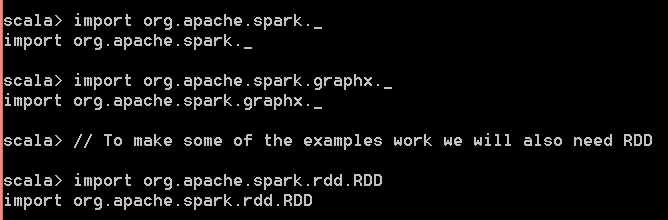
GRAPH X

GraphX ​​adalah komponen baru dalam Spark untuk grafik dan perhitungan grafik-paralel. Pada tingkat tinggi, GraphX ​​memperluas Spark [RDD](https://spark.apache.org/docs/latest/api/scala/index.html" \l "org.apache.spark.rdd.RDD)dengan mengenalkan abstraksi [Grafik](https://spark.apache.org/docs/latest/graphx-programming-guide.html" \l "property_graph) baru : sebuah multigraph yang diarahkan dengan properti yang melekat pada masing-masing sudut dan tepi. Untuk mendukung perhitungan grafik, GraphX ​​memperlihatkan satu set operator fundamental (misalnya [subgraf](https://spark.apache.org/docs/latest/graphx-programming-guide.html" \l "structural_operators) , [joinVertices](https://spark.apache.org/docs/latest/graphx-programming-guide.html" \l "join_operators) , dan[aggregateMessages](https://spark.apache.org/docs/latest/graphx-programming-guide.html#aggregateMessages) ) serta varian [Optimal Pregel](https://spark.apache.org/docs/latest/graphx-programming-guide.html#pregel) API yang dioptimalkan . Selain itu, GraphX ​​mencakup kumpulan [algoritma](https://spark.apache.org/docs/latest/graphx-programming-guide.html" \l "graph_algorithms) grafik dan[pembangun yang](https://spark.apache.org/docs/latest/graphx-programming-guide.html#graph_builders) berkembang untuk menyederhanakan tugas analisis grafik.

* Untuk memulai  pertama-tama Anda perlu mengimpor Spark dan GraphX ​​ke dalam proyek Anda, sebagai berikut:
* **import** **org.apache.spark.\_**
* **import** **org.apache.spark.graphx.\_**
* *// To make some of the examples work we will also need RDD*
* **import** **org.apache.spark.rdd.RDD**

Jika Anda tidak menggunakan Spark shell Anda juga memerlukan a SparkContext. Untuk mempelajari lebih lanjut tentang memulai dengan Spark,

Hasil:



* grafik property

The grafik properti adalah Multigraph diarahkan dengan yang ditetapkan pengguna benda yang melekat pada setiap sudut dan tepi. Multigraph yang terarah adalah graf berarah dengan beberapa tepi paralel yang berpotensi berbagi simpul sumber dan tujuan yang sama. Kemampuan untuk mendukung tepi paralel menyederhanakan skenario pemodelan dimana ada banyak hubungan (misalnya rekan kerja dan teman) di antara simpul yang sama. Setiap simpul dikunci oleh pengenal panjang 64-bit yang unik ( VertexId). GraphX ​​tidak memaksakan batasan pemesanan pada pengenal simpul. Demikian pula, tepi memiliki pengenal simpul sumber dan tujuan yang sesuai. Grafik properti diberi parameter pada tipe vertex ( VD) dan edge ( ED). Ini adalah jenis objek yang terkait dengan masing-masing sudut dan tepi masing-masing.

Dalam beberapa kasus, mungkin diinginkan memiliki simpul dengan jenis properti yang berbeda pada grafik yang sama. Hal ini dapat dicapai melalui pewarisan. Misalnya untuk model pengguna dan produk sebagai grafik bipartit kita bisa melakukan hal berikut:

**class** **VertexProperty**()

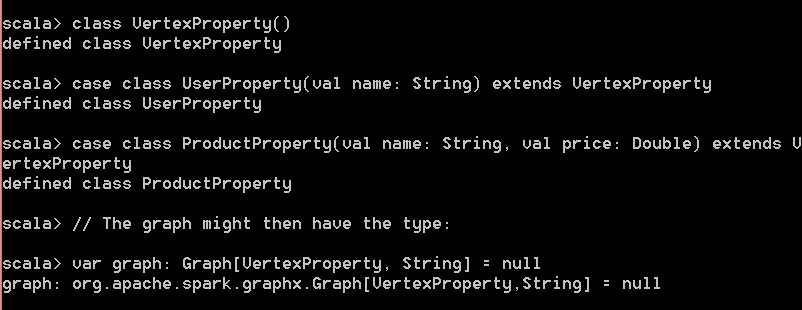
**case** **class** **UserProperty**(**val** name**:** String) **extends** **VertexProperty**

**case** **class** **ProductProperty**(**val** name**:** String, **val** price**:** Double) **extends** **VertexProperty**

*// The graph might then have the type:*

**var** graph**:** Graph[VertexProperty, String] **=** **null**

hasil:



Seperti RDD, grafik properti tidak berubah, terdistribusi, dan toleransi kesalahan. Perubahan pada nilai atau struktur grafik dilakukan dengan menghasilkan grafik baru dengan perubahan yang diinginkan. Perhatikan bahwa sebagian besar grafik asli (yaitu, struktur, atribut, dan indeks yang tidak terpengaruh) digunakan kembali dalam grafik baru yang mengurangi biaya struktur data yang inheren ini. Grafik dipartisi di seluruh pelaksana dengan menggunakan serangkaian heuristik partisi verteks. Seperti RDD, setiap partisi grafik dapat diciptakan kembali pada mesin yang berbeda jika terjadi kegagalan.

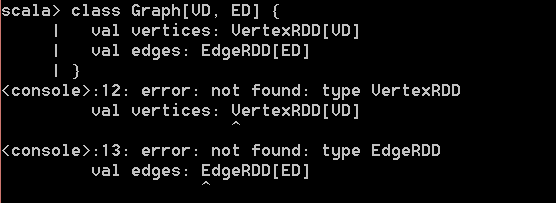
Logikanya grafik properti sesuai dengan sepasang koleksi yang diketik (RDD) yang mengkodekan properti untuk setiap simpul dan tepi. Sebagai konsekuensinya, kelas grafik berisi anggota untuk mengakses simpul dan tepi grafik:

**class** **Graph**[VD, ED] {

**val** vertices**:** VertexRDD[VD]

**val** edges**:** EdgeRDD[ED]

}

Hasil:  


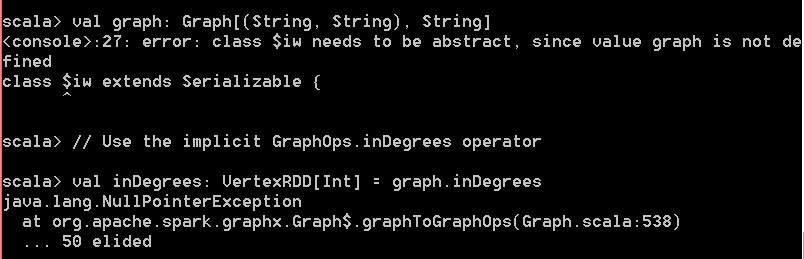
* operator grafik

Sama seperti RDD memiliki operasi dasar seperti map,, filterdan reduceByKey, grafik properti juga memiliki kumpulan operator dasar yang mengambil fungsi yang ditentukan pengguna dan menghasilkan grafik baru dengan sifat dan struktur yang berubah. Operator inti yang memiliki implementasi optimal didefinisikan Graphdan operator yang sesuai yang dinyatakan sebagai komposisi dari operator inti ditetapkan GraphOps. Namun, berkat Scala, implisit operator GraphOpssecara otomatis tersedia sebagai anggota Graph. Sebagai contoh, kita dapat menghitung in-degree dari setiap vertex (didefinisikan dalam GraphOps) sebagai berikut:

**val** graph**:** Graph[(String, String), String]

*// Use the implicit GraphOps.inDegrees operator*

**val** inDegrees**:** VertexRDD[Int] **=** graph.inDegrees



Alasan untuk membedakan antara operasi grafik inti dan GraphOpsuntuk dapat mendukung representasi grafik yang berbeda di masa depan. Setiap representasi grafik harus menyediakan implementasi operasi inti dan menggunakan kembali banyak operasi yang berguna yang didefinisikan GraphOps.