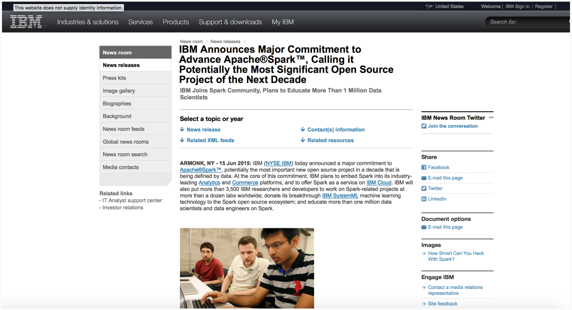
Introduction to Spark

Louis Jachiet

Spark Motivation

Apache Spark



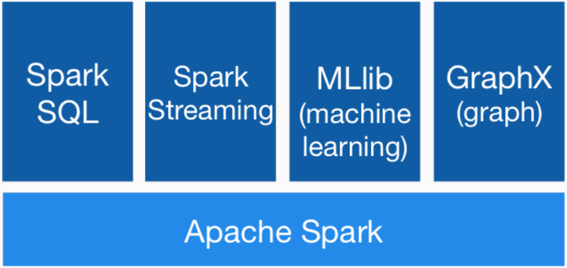
IBM and Apache Spark

What is Apache Spark

Apache Spark is a fast and general engine for large-scale data processing.

- **Speed**: Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.
- **Ease of Use**: Write applications quickly in Java, Scala, Python, R.
- **Generality**: Combine SQL, streaming, and complex analytics.
- **Runs Everywhere**: Spark runs on Hadoop, Mesos, standalone, or in the cloud.

Spark Ecosystem



Spark API

```
text_file = spark.textFile("hdfs://...")

text_file.flatMap(lambda line: line.split())
     .map(lambda word: (word, 1))
     .reduceByKey(lambda a, b: a+b)
```

Word count in Spark's Python API

Spark API

```scala
val f = sc.textFile("hdfs://...")

val wc = f.flatMap(l => l.split(" "))
          .map(word => (word, 1))
          .reduceByKey(_ + _)
```

Word count in Spark's Scala API

Apache Spark

Apache Spark Project

- Spark started as a research project at UC Berkeley
  - Matei Zaharia created Spark during his PhD
  - Ion Stoica was his advisor
- DataBricks is the Spark start-up, that has raised $46 million

$\Rightarrow$ *Now an Apache project*

Resilient Distributed Datasets (RDDs)

- An RDD is a fault-tolerant collection of elements that can be operated on in parallel.
- RDDs are created :
  - parallelizing an existing collection in your driver program, or
  - referencing a dataset in an external storage system

Spark API: Parallel Collections

```
data = [1, 2, 3, 4, 5]
distData = sc.parallelize(data)
```

Spark's Python API

Spark API: Parallel Collections

```scala
val data = Array(1, 2, 3, 4, 5)
val distData = sc.parallelize(data)
```

Spark's Scala API

Spark API: Parallel Collections

```java
List<Integer> data = Arrays.asList(1, 2, 3, 4, 5);
JavaRDD<Integer> distData = sc.parallelize(data);
```

Spark's Java API

Spark API: External Datasets

```
>>> distFile = sc.textFile("data.txt")
PythonRDD[60] at RDD at PythonRDD.scala:53
```

Spark's Python API

```
scala> val distFile = sc.textFile("data.txt")
distFile: RDD[String] = MappedRDD@1d4cee08
```

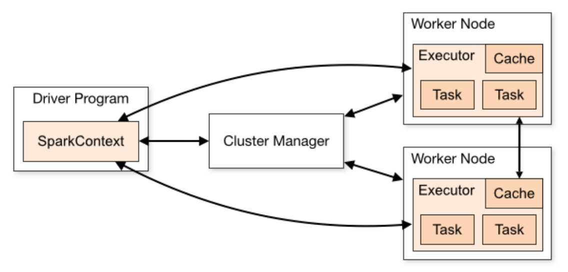
Spark's Scala API

Spark API: External Datasets

```java
JavaRDD<String> distFile = sc.textFile("data.txt");
```

Spark's Java API

Spark Cluster

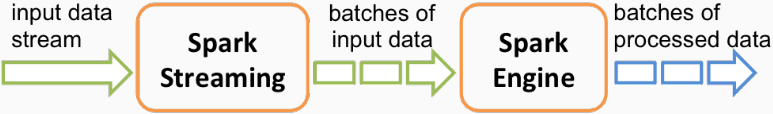


Cluster Components

Spark Cluster

- Spark is agnostic to the underlying cluster manager.
- The spark driver is the program that declares the transformations and actions on RDDs of data and submits such requests to the master.
- Each application gets its own executor processes, which stay up for the duration of the whole application and run tasks in multiple threads. Each driver schedules its own tasks.
- The drivers must listen for and accept incoming connections from its executors throughout its lifetime
- Because the driver schedules tasks on the cluster, it should be

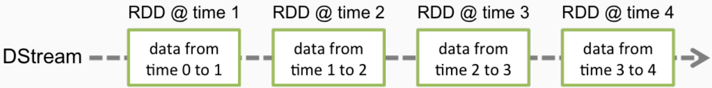
Apache Spark Streaming



Spark Streaming is an extension of Spark that allows processing data stream using micro-batches of data.

Discretized Streams (DStreams)

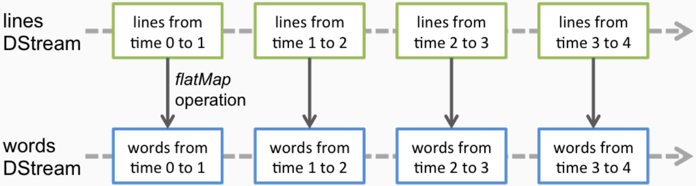
- Discretized Stream or DStream represents a continuous stream of data,
  - either the input data stream received from source, or
  - the processed data stream generated by transforming the input stream.
- Internally, a DStream is represented by a continuous series of RDDs



Discretized Streams (DStreams)

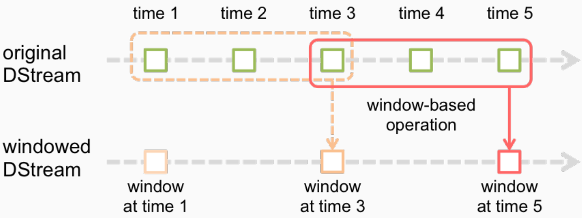
Spark Streaming

- Any operation applied on a DStream translates to operations on the underlying RDDs.



Discretized Streams (DStreams)

- Spark Streaming provides windowed computations, which allow transformations over a sliding window of data.



Spark Streaming

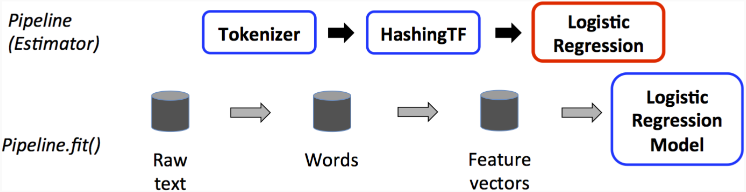
```scala
val conf = new SparkConf().setMaster("local[2]").setAppName("WCount")
val ssc = new StreamingContext(conf, Seconds(1))

// Create a DStream that will connect to hostname:port, like localhost:9999
val lines = ssc.socketTextStream("localhost", 9999)

// Split each line into words
val words = lines.flatMap(_.split(" "))

// Count each word in each batch
val pairs = words.map(word => (word, 1))
val wordCounts = pairs.reduceByKey(_ + _)

// Print the first ten elements of each RDD generated in this DStream to the console
wordCounts.print()

ssc.start()             // Start the computation
ssc.awaitTermination()  // Wait for the computation to terminate
```

Spark SQL and DataFrames

- Spark SQL is a Spark module for structured data processing.
- It provides a programming abstraction called DataFrames and can also act as distributed SQL query engine.
- A DataFrame is a distributed collection of data organized into named columns. It is conceptually equivalent to a table in a relational database .

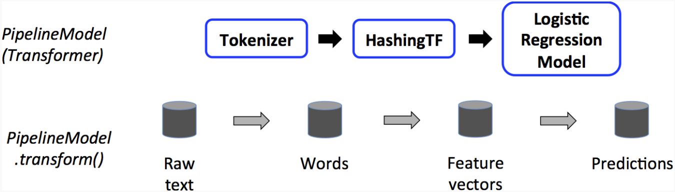
Spark Machine Learning Libraries

- MLLib contains the original API built on top of RDDs.
- spark.ml provides higher-level API built on top of DataFrames for constructing ML pipelines.



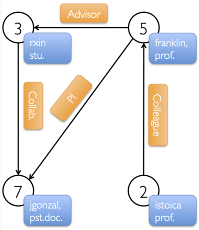
Spark Machine Learning Libraries

- MLLib contains the original API built on top of RDDs.
- spark.ml provides higher-level API built on top of DataFrames for constructing ML pipelines.



- GraphX optimizes the representation of vertex and edge types when they are primitive data types
- The property graph is a directed multigraph with user defined objects attached to each vertex and edge.

Property Graph



Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

Spark GraphX

```scala
// Assume the SparkContext has already been constructed
val sc: SparkContext
// Create an RDD for the vertices
val users: RDD[(VertexId, (String, String))] =
  sc.parallelize(Array((3L, ("rxin", "student")), (7L, ("jgonzal", "postdoc")),
                       (5L, ("franklin", "prof")), (2L, ("istoica", "prof"))))
// Create an RDD for edges
val relationships: RDD[Edge[String]] =
  sc.parallelize(Array(Edge(3L, 7L, "collab"),    Edge(5L, 3L, "advisor"),
                       Edge(2L, 5L, "colleague"), Edge(5L, 7L, "pi")))
// Define a default user in case there are relationship with missing user
val defaultUser = ("John Doe", "Missing")
// Build the initial Graph
val graph = Graph(users, relationships, defaultUser)
```

Apache Spark Summary

Apache Spark is a fast and general engine for large-scale data processing.

- **Speed**: Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.
- **Ease of Use**: Write applications quickly in Java, Scala, Python, R.
- **Generality**: Combine SQL, streaming, and complex analytics.
- **Runs Everywhere**: Spark runs on Hadoop, Mesos, standalone, or in the cloud.

Spark Resilient Distributed Datasets

Resilient Distributed Datasets (RDD)

Key ideas for Spark

1. Keep a trace of how data was constructed

Resilient Distributed Datasets (RDD)

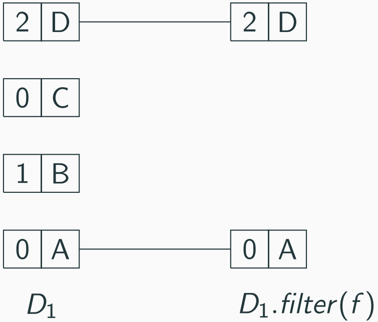
Key ideas for Spark

1. Keep a trace of how data was constructed
2. Computation failure is rare

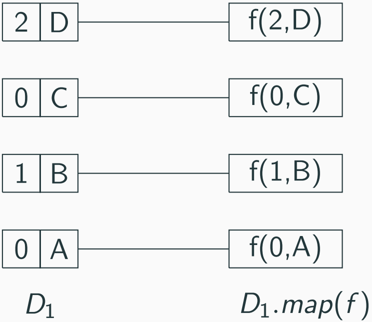
Key ideas for Spark

1. Keep a trace of how data was constructed
2. Computation failure is rare
3. Provide high(er)-level constructs and interactivity

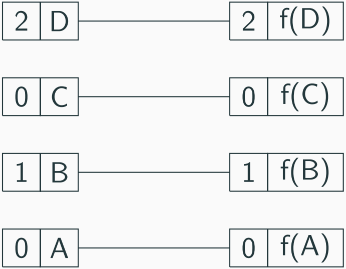
Simple operations on RDD



Simple operations on RDD

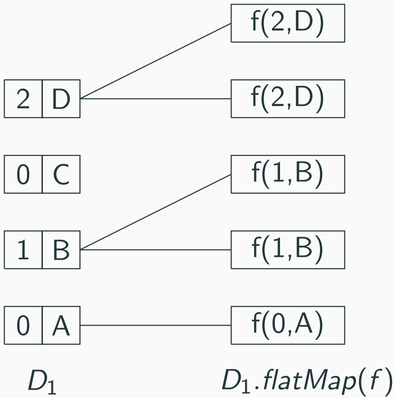


Simple operations on RDD



D_1 $D_1.mapValues(f)$

Simple operations on RDD



D_1		$D_1.groupByKey()$	
2	D	3	$\{\eta\}$
0	C		
1	B	1	$\{\beta\}$
0	A	0	$\{\alpha, \gamma\}$

Simple operations on RDD

D_1		D_2	
2	D	3	η
0	C	0	γ
1	B	1	β
0	A	0	α

Simple operations on RDD

2 | D	3 | η
0 | C	0 | γ
1 | B	1 | β
0 | A	0 | α
D_1	D_2

$$D_1.union(D_2)$$

2	D
0	C
1	B
0	A

D_1

3	η
0	γ
1	β
0	α

D_2

1	B	β
0	C	α
1	A	γ
0	A	α
0	C	γ

$D_1.join(D_2)$

Simple operations on RDD

D_1		D_2		$D_1.coGroup2(D_2)$	
2	D	3	η	2	$\{D\}, \emptyset$
0	C	0	γ	3	$\emptyset, \{\eta\}$
1	B	1	β	1	$\{B\}, \{\beta\}$
0	A	0	α	0	$\{A, C\}, \{\alpha, \gamma\}$

Practical Spark

Spark RDD API (extract)

filter	limits the number of records
map	transform records
mapValues	transforms only the value
flatMap	maps each record to 0, 1 or more elements
reduce	combines all elements
fold	combines all elements with an initial value
aggregate	fold+reduce
distinct	eliminates duplicates

Manipulating RDDs

Spark RDD API (extract)

groupByKey	corresponds to a shuffle
reduceByKey(f)	reduce for each key
foldByKey(f)	fold for each key
keyBy(f)	create pair-RDD from RDD

Manipulating pair-RDDs

Spark RDD API (extract)

cartesian	build the cartesian product
join	behaves like SQL join (supposes two pair-RDDs)
union	takes the union of two RDDs
intersection	takes the union of two RDDs
substract	takes the union of two RDDs
coGroup	generalized groupByKey (supposes two pair-RDDs)

Combining several RDDs

Spark RDD API (extract)

textFile(*path*)	creates an RDD with an item per line
saveAsTextFile(*path*)	saves an RDD with an item per line
sortWith(f)	sort according to comparison f

Tools for RDDs

take(n)	retrieves the n first elements
collect()	retrieves whole RDD
count()	counts the number of items
fold / reduce / aggregate	
foreach	apply a function on each element

Actions on RDDs

Spark API: RDD Operations

```
lines = sc.textFile("data.txt")
lineLengths = lines.map(lambda s: len(s))
totalLength = lineLengths.reduce(lambda a, b: a + b)
```

Spark's Python API

Spark API: RDD Operations

```scala
val lines = sc.textFile("data.txt")
val lineLengths = lines.map(s => s.length)
val totalLength = lineLengths.reduce((a, b) => a + b)
```

Spark's Scala API

Spark API: RDD Operations

```
JavaRDD<String> lines = sc.textFile("data.txt");
JavaRDD<Integer> lineLengths = lines.map(s -> s.length());
int totalLength = lineLengths.reduce((a, b) -> a + b);
```

Spark's Java API

```python
lines = sc.textFile("data.txt")
pairs = lines.map(lambda s: (s, 1))
counts = pairs.reduceByKey(lambda a, b: a + b)
```

Spark's Python API

Spark API: Working with Key-Value Pairs

```
val lines = sc.textFile("data.txt")
val pairs = lines.map(s => (s, 1))
val counts = pairs.reduceByKey((a, b) => a + b)
```

Spark's Scala API

Spark API: Working with Key-Value Pairs

```
JavaRDD<String> lines = sc.textFile("data.txt");
JavaPairRDD<String, Integer> pairs =
    lines.mapToPair(s -> new Tuple2(s, 1));
JavaPairRDD<String, Integer> counts =
    pairs.reduceByKey((a, b) -> a + b);
```

Spark's Java API

Exercise revisited (easy)

Input

You are given a list of pairs (k_i, v_i) where k_i is a string and v_i an integer.

Problem

Compute the average value for each key.

Example

INPUT	
A	42
B	17
A	12
B	99

OUTPUT	
A	$\frac{42+12}{2} = 27$
B	$\frac{17+99}{2} = 58$

Exercise revisited (medium)

Input

You are given two lists of items.

Problem

Compute the list of item appearing in the first one but not in the second.

Example

INPUT 1
A
B
C

INPUT2
A
C
E

OUTPUT
B

Exercise revisited (hard)

Input

You are given the Twitter following list: each record is a pair (A_i, B_i) indicating that account A_i follows B_i.

Problem

Compute the accounts that have more followers than followees.

Example

INPUT	
A	B
A	D
B	C
B	D
C	E

OUTPUT
E
D
C

Exercise revisited (hardest)

Input

You are given the Twitter following list: each record is a pair (A_i, L_i) indicating that account A_i follows the accounts in the list L_i .

Problem

Compute for each account A the list of accounts that are followed by an account followed by A.

Example

INPUT	
A	B,D
B	C,D
C	E

OUTPUT	
A	C,D
B	E

Iterations

Iteration

Spark is especially competitive for jobs requiring long chain of individual jobs.

Iteration

Spark is especially competitive for jobs requiring long chain of individual jobs.

Such jobs are often required by data mining algorithm (e.g. the gradient descent for logistic regression).

Logistic regression

```python
# Compute logistic regression gradient for a matrix of data points
def gradient(matrix, w):
    Y = matrix[:, 0]    # point labels (first column of input file)
    X = matrix[:, 1:]   # point coordinates
    # For each point (x, y), compute gradient function, then sum these up
    return ((1.0 / (1.0 + np.exp(-Y * X.dot(w))) - 1.0) * Y * X.T).sum(1)

def add(x, y):
     x += y
     return x

for i in range(iterations):
     print("On iteration %i" % (i + 1))
     w -= points.map(lambda m: gradient(m, w)).reduce(add)
```

From Spark: examples/src/main/python/logistic_regression.py

Caching and persistence

Context

RDD materialization are only triggered by an *action*.

Context

RDD materialization are only triggered by an *action*.

Spark caches by default the materialization, but the user can specify the caching.

Caching

Type	Space	CPU
MEMORY_ONLY*	high	low
MEMORY_ONLY_SER	low	high
MEMORY_AND_DISK	high	med
MEMORY_AND_DISK_SER	low	high
DISK_ONLY	low	high

Logistic regression

```python
# Compute logistic regression gradient for a matrix of data points
def gradient(matrix, w):
    Y = matrix[:, 0]    # point labels (first column of input file)
    X = matrix[:, 1:]   # point coordinates
    # For each point (x, y), compute gradient function, then sum these up
    return ((1.0 / (1.0 + np.exp(-Y * X.dot(w))) - 1.0) * Y * X.T).sum(1)

def add(x, y):
     x += y
     return x

for i in range(iterations):
     print("On iteration %i" % (i + 1))
     w -= points.map(lambda m: gradient(m, w)).reduce(add)
```

From Spark: examples/src/main/python/logistic_regression.py

Shared Variables

```
>>> broadcastVar = sc.broadcast([1, 2, 3])

>>> broadcastVar.value
[1, 2, 3]
```

Spark's Python API

Spark API: Shared Variables

```
scala> val broadcastVar = sc.broadcast(Array(1, 2, 3))

scala> broadcastVar.value
res0: Array[Int] = Array(1, 2, 3)
```

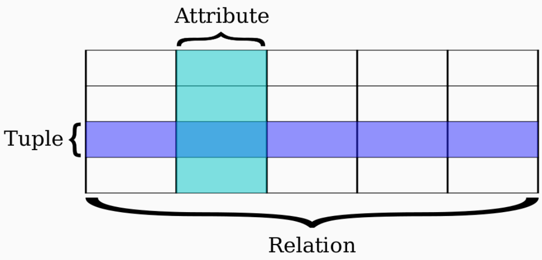
Spark's Scala API

```
Broadcast<int[]> broadcastVar =
      sc.broadcast(new int[] {1, 2, 3});

broadcastVar.value();
// returns [1, 2, 3]
```

Spark's Java API

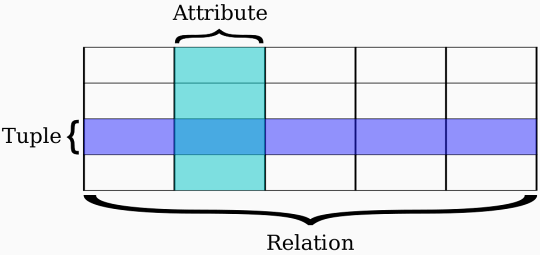
Spark Dataframes

SQL model



SQL tables

SQL model



SQL tables

Dataframes are basically RDD with an explicit schema but:

1. untyped data
2. can be optimized

Creating Dataframes From RDDs 1/3

```scala
val movies = ... // of type RDD[(int,string,string)]
 ...
val dfMovies = movies.toDF("movieId","title","genre")
```

Creating Dataframes From RDDs 2/3

```scala
class Movie(movieId: Int, title: String,genre: String)
...
val movies = ... // of type RDD[Movie]
 ...
val dfMovies = movies.toDF
```

Creating Dataframes From RDDs 3/3

```
val myRDD = something //
...
val mySchema = StructType(List(
     StructField("number", IntegerType, true),
     StructField("word", StringType, true)))
...
val myDF = spark.createDataFrame(myRDD,mySchema)
```

Creating Dataframes From Files

Spark can read from:

1. CSV
2. JSON
3. Parquet
4. *etc.*

```
val myDF = spark.read.format("csv").
                 option("header","true").
                 load("/datasets/movie_small/ratings.csv")
```

Creating Dataframes From Files

Spark can read from:

1. CSV
2. JSON
3. Parquet
4. *etc.*

```scala
import org.apache.spark.sql.types.{StructType, StructField, FloatType,
      IntegerType, LongType};
val mySchema = StructType(List(StructField("userId", IntegerType, true),
                 StructField("movieId", IntegerType, true),
                 StructField("rating", FloatType, true),
                 StructField("timestamp", LongType, true)))
val myDF = spark.read.format("csv").option("header","true").
                 schema(mySchema).
                 load("/datasets/movie_small/ratings.csv")
```

Operations on DataFrames

select	projection on some columns
agg	aggregation
groupBy	use in conjunction with agg
join	inner join
filter	filter some columns
limit	equivalent to take(n)
orderBy	sort by a given column
where	condition on join
union	union
show	print 20 first entries
printSchema	print schema
as	name table
drop	remove records with NULL
fill	replace NULL with value

Example

```
val schema2 = StructType(List(
    StructField("movieId",IntegerType),
    StructField("title",StringType)))
    val movieDF = spark.read.format("csv").
                  option("header","true").
                  schema(schema2).
                  load("/datasets/movie_small/movies.csv")
    myDF2.join(movieDF,
                myDF2("movieId")===movieDF("movieId")).
    show()
```

Exercise revisited (easy)

Input

You are given a list of pairs (k_i, v_i) where k_i is a string and v_i an integer.

Problem

Compute the average value for each key.

Example

INPUT	
A	42
B	17
A	12
B	99

OUTPUT	
A	$\frac{42+12}{2} = 27$
B	$\frac{17+99}{2} = 58$

Exercise revisited (medium)

Input

You are given two lists of items.

Problem

Compute the list of item appearing in the first one but not in the second.

Example

INPUT 1
A
B
C

INPUT2
A
C
E

OUTPUT
B

Exercise revisited (hard)

Input

You are given the Twitter following list: each record is a pair (A_i, B_i) indicating that account A_i follows B_i.

Problem

Compute the accounts that have more followers than followees.

Example

INPUT	
A	B
A	D
B	C
B	D
C	E

OUTPUT
E
D
C

Exercise revisited (hardest)

Input

You are given the Twitter following list: each record is a pair (A_i, L_i) indicating that account A_i follows the accounts in the list L_i .

Problem

Compute for each account A the list of accounts that are followed by an account followed by A.

Example

INPUT	
A	B,D
B	C,D
C	E

OUTPUT	
A	C,D
B	E

Datasets

Datasets

Dataframes can be generalized into *Datasets*, giving the best of both worlds.

Spark SQL

Use any SQL command

Example!

Exercise revisited (easy)

Input

You are given a list of pairs (k_i, v_i) where k_i is a string and v_i an integer.

Problem

Compute the average value for each key.

Example

INPUT	
A	42
B	17
A	12
B	99

OUTPUT	
A	$\frac{42+12}{2} = 27$
B	$\frac{17+99}{2} = 58$

Exercise revisited (medium)

Input

You are given two lists of items.

Problem

Compute the list of item appearing in the first one but not in the second.

Example

INPUT 1
A
B
C

INPUT2
A
C
E

OUTPUT
B

Exercise revisited (hard)

Input

You are given the Twitter following list: each record is a pair (A_i, B_i) indicating that account A_i follows B_i.

Problem

Compute the accounts that have more followers than followees.

Example

INPUT	
A	B
A	D
B	C
B	D
C	E

OUTPUT
E
D
C

Exercise revisited (hardest)

Input

You are given the Twitter following list: each record is a pair (A_i, L_i) indicating that account A_i follows the accounts in the list L_i .

Problem

Compute for each account A the list of accounts that are followed by an account followed by A.

Example

INPUT	
A	B,D
B	C,D
C	E

OUTPUT	
A	C,D
B	E

Playing With The Movie Lens Dataset