

1 MapReduce

For each problem below, write pseudocode to complete the implementations. Tips:

- The input to each MapReduce job is given by the signature of `map()`.
- `emit(key k, value v)` outputs the key-value pair `(k, v)`.
- `for var in list` can be used to iterate through `Iterables` or you can call the `hasNext()` and `next()` functions.
- Usable data types: `int`, `float`, `String`. You may also use lists and custom data types composed of the aforementioned types.
- `intersection(list1, list2)` returns a list of the intersection of `list1`, `list2`.

1.1 Given a set of coins and each coin's owner, compute the number of coins of each denomination that a person has.

Declare any custom data types here:

CoinPair:

```
String person
String coinType
```

```
1 map(_____, _____):      1 reduce(_____, _____):
```

1.2 Using the output of the first MapReduce, compute each person's amount of money. `valueOfCoin(String coinType)` returns a float corresponding to the dollar value of the coin.

```
1 map(_____, _____):      1 reduce(_____, _____):
```

2 Spark

Resilient Distributed Datasets (RDD) are the primary abstraction of a distributed collection of items

Transforms $RDD \rightarrow RDD$

`map(f)` Return a new dataset formed by calling f on each source element.

`flatMap(f)` Similar to `map`, but each input item can be mapped to 0 or more output items (so f should return a sequence rather than a single item).

`reduceByKey(f)` When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function f , which must be of type $(V, V) \rightarrow V$.

Actions $RDD \rightarrow Value$

`reduce(f)` Aggregate the elements of the dataset *regardless of keys* using a function f .

Call `sc.parallelize(data)` to parallelize a Python collection, `data`.

- 2.1 Given a set of coins and each coin's owner, compute the number of coins of each denomination that a person has. Then, using the output of the first result, compute each person's amount of money. Assume `valueOfCoin(coinType)` is defined and returns the dollar value of the coin.

The type of `coinPairs` is a list of `(person, coinType)` pairs.

```
1 coinData = sc.parallelize(coinPairs)
```

3 Warehouse-Scale Computing

Sources speculate Google has over 1 million servers. Assume each of the 1 million servers draw an average of 200W, the PUE is 1.5, and that Google pays an average of 6 cents per kilowatt-hour for datacenter electricity.

- 3.1 Estimate Google's annual power bill for its datacenters.
- 3.2 Google reduced the PUE of a 50,000-machine datacenter from 1.5 to 1.25 without decreasing the power supplied to the servers. What's the cost savings per year?

4 MapReduce/Spark Practice: Optimize Your GPA

4.1 Given the student's name and course taken, output their name and total GPA.

Declare any custom data types here:

CourseData:

```
int courseID
float studentGrade // a number from 0-4
```

```
1 map(_____, _____):           1 reduce(_____, _____):
```

4.2 Solve the problem above using Spark.

The type of `students` is a list of `(studentName, courseData)` pairs.

```
1 studentsData = sc.parallelize(students)
2 out = studentsData.map(lambda (k, v): (k, (v.studentGrade, _____)))
```

5 MapReduce/Spark Practice: Optimize the Friend Zone

5.1 Given a person's unique int ID and a list of the IDs of their friends, compute the list of mutual friends between each pair of friends in a social network. You have access to the `intersection` function, which takes in two lists finds the set of elements that appear in both lists.

Declare any custom data types here:

FriendPair:

```
int friendOne
int friendTwo
```

```
1 map(int personID, list<int> friendIDs):           1 reduce(_____, _____):
```

5.2 Solve the problem above using Spark.

The type of `persons` is a list of `(personID, list(friendID))` pairs.

```
1 def genFriendPairAndValue(pID, fIDs):
2     return [(pID, fID), fIDs] if pID < fID else (fID, pID) for fID in fIDs]
3
4 def intersection(l1, l2):
5     return [x for x in b1 if x in b2]
6
7 personsData = sc.parallelize(persons)
```