A problem: find people you may know

A Social Network

- Task: suggest friends
- There are several factors to consider
- A very important one is by number of mutual friends
- How to compute a list of suggestions?
- Consider a simple criterion:

If X and Y have at least 3 mutual friends then Y is a suggested friend of X

The Data

- For every member, there is a list of friends

A	→	B	C	L	...
B	→	A	G	K	...
C	→	A	G	M	...

Naïve approach:

- For every friend X of A $O(n)$
  - For every friend Y of X $O(n)$
    - Compute number of mutual friends between Y and A
    - Intersect the friend-list of Y and friend-list of A $O(n)$

$O(n^3)$ per member. Total $O(N\,n^3)$!!

N members.
Every member has n friends.
$N \gg n$.

Not a good approach!

A better method

A	→	B	C	L	...
B	→	A	G	K	...
C	→	A	G	M	...

- For each friend X of A $O(n)$
  - For each friend Y of X $O(n)$
    - Write: (A, Y, X) $O(n^2)$
- What do we get?
- Now sort by first two entries
  - All the (A,G)'s are together
- For each (A, Y), count how many times it occurs
  - Number of mutual friends between A and Y

(A, A, B)
(A, G, B)
(A, K, B)
...
(A, G, C)
...
(A, G, P)

$O(n^2 \log n)$

...
...
(A, G, B)
(A, G, C)
(A, G, P)
...
...

$O(n^2)$

★ $O(n^2 \log n)$ per member ★

Using MapReduce

Map:

- For each member X, fetch his/her friendlist $Y_1, Y_2, \ldots, Y_n$
  - For each pair (Y_1, Y_2), emit the key value pair $\{(Y_1, Y_2), X\}$

C → A G M ...

Map

Key	Value
(A,G)	C
(A,M)	C
(G,M)	C
...	...

B → A G K ...

Map

Key	Value
(A,G)	B
(A,K)	B
(G,K)	B
...	...

Note: If C is a friend of A, implies A is also a friend of C.
eg. Facebook friend list.

Using MapReduce

Key	Value
(A,G)	C
(A,M)	C
(G,M)	C
...	...
(A,G)	B
(A,K)	B
(G,K)	B
...	...

Shuffle and sort

Key	Value
(A,G)	B
(A,G)	C
(A,G)	P
...	...
(S,T)	Q
(S,T)	V
(S,T)	P
...	...

→ count (A,G) → No. of mutuals ← count (S,T)

Complexity :

X X X

Spark (pyspark)

```
from pyspark import SparkContext, SparkConf
sc = SparkContext.getOrCreate()
```

Resilient Distributed Dataset (RDD)

- It is a core datastructure in Apache Spark.
- 2. RDDs are immutable, partitioned collections of data that can be processed in parallel across a cluster.

3. RDDs provide a simple programming model for distributed data processing, allowing developers to perform operations like map, reduce, and filter on large datasets.

- If we pass, say 100 lines
- Spark stores chunks of lines in partitions
  - AND NOT in parts of lines.

NOTE: Size of line should not be greater than block size.

- No. of partitions can be controlled :

  dist_data = sc.parallelize (data, 4)
  ↳ #partions

- .saveasTextFile ("location")
  ↳ saves result as text file onto memory (local file system)

- **Lambda Functions / Anonymous function:**

  eg: add_two = lambda a, b : a+b
  print (add_two (2, 3))

  This is similar to :

  def add_two (a, b) :
  return (a+b)

- **Map :**

  Map takes a func^n & list as args and applies the functions to all the elements of list.

  eg.:
  ```
  def square (a) :
      return a**2
  sq_map = list (map (square, data))
  ```
  ↳ where data is a list of numbers.

- Map using lambda :

  data_sq = list (map (lambda a : a*a, data))

* Map takes a function and a list and applies the function to the list "element wise"

- **Reduce :**
  ↳
  ```
  from functools import reduce
  data_sum = reduce (lambda a,b : a+b, data)
  print (data_sum)
  ```

- **Spark RDD basic operations :**

  ① dist_data.collect ( )
  ↳ returns all elements of the RDD as an array.
  NOTE: Not recommended for large RDDs (Big Data)

  ② dist_data.first ( )
  returns first element

  ③ dist_data.takeSample (False, 5)
  False ⇒ without replacement
  5 ⇒ Number of elements

  ④ dist_data.count ( )
  returns length / size of array (RDD)

* **RDD from external files :**

eg.: `lines = sc.textFile("path_to_file")`

`lines.takeSample(False, 5)`

returns 5 lines (may contain '' (blank lines))

> **Word Count example :**

`words = list(map(lambda line: line.split(" "), lines))`

`words.takeSample(False, 10)`

Note: For each line `.split(" ")` will convert the line into a list of words. This will return a list of lists, as we have many lines.

Solution:

Use `flatMap( )` instead of `map( )`.

Remember :

lines was an RDD
words is also an RDD

Next, we perform the actual map task of MapReduce i.e emit (w, 1) for every w in words.

`keyVal = list(map(lambda w: (w, 1)))`

NOTE: keyVal is also an RDD.

Next, Reduce Task of MapReduce

`counts = keyVal.reduceByKey(lambda m,n: m+n)`

counts is also an RDD

`counts.takeSample(False, 5)`