BIG DATA MANAGEMENT

CE/CZ4123

DISTRIBUTED SYSTEMS AND MAP-REDUCE (MORE EXAMPLES)

MapReduce:
A general distributed paradigm

In this lecture, we will see more complicated examples of MapReduce based algorithms.

- ❑ Table Join
- ❑ Shortest Path Computation
- ❑ PageRanks

NOTES ON PSEUDOCODE FOR MAPREDUCE ALGORITHMS

- The pesudocode focuses on “thinking in MapReduce”, and so it is okay to use any understandable syntax (C-like or Java-like).
- For presenting complicated MapReduce Algorithms, we can use more complicated object class (data structure) than “String” for both key and value. Then we should describe the composition of the complicated data structure as well.

EXAMPLE: JOINING TWO TABLES

Primary key

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

Foreign key

C	D	E
c1	d1	e1
c1	d2	e2
c2	d3	e3
c3	d4	e4
c3	d5	e5

Map Input

Key: Line number
Value: A;B;C

Key: Line number
Value: C;D;E

JOINING TWO TABLES

Primary key

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

Foreign key

C	D	E
c1	d1	e1
c1	d2	e2
c2	d3	e3
c3	d4	e4
c3	d5	e5

Map Input

Key: Line number
Value: A;B;C

Key: Line number
Value: C;D;E

- If the tuples in two tables are in the same file, then we need to know the size of the 1st table to classify the tuples.

JOINING TWO TABLES

Primary key

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

Foreign key

C	D	E
c1	d1	e1
c1	d2	e2
c2	d3	e3
c3	d4	e4
c3	d5	e5

Map Output

Key: C
Value: T1;A;B

Key: C
Value: T2;D;E

JOINING TWO TABLES

Primary key

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

Foreign key

C	D	E
c1	d1	e1
c1	d2	e2
c2	d3	e3
c3	d4	e4
c3	d5	e5

Reduce Input

Key: C
Value: {T1;A;B, T2;D;E}

Key: c1
Value: {T1;a1;b1, T2;d1;e1, T2;d2;e2}

Key: c2
Value: {T1;a2;b2, T2;d3;e3}

Key: c3
Value: {T1;a3;b3, T2;d4;e4, T2;d5;e5}

JOINING TWO TABLES

Primary key

A	B	C
a1	b1	c1
a2	b2	c2
a3	b3	c3
a4	b4	c4
a5	b5	c5

Foreign key

C	D	E
c1	d1	e1
c1	d2	e2
c2	d3	e3
c3	d4	e4
c3	d5	e5

Reduce

Key: c1
Value: {T1;a1;b1, T2;d1;e1, T2;d2;e2} → (T1;a1;b1, T2;d1;e1)
(T1;a1;b1, T2;d2;e2)

↘ (a1;b1;c1;d1;e1)
(a1;b1;c1;d2;e2)

A FEW TIPS

- It is crucial to determine which attributes/features should be aggregated on → Intermediate key

- The intermediate value can be complex; it can include a lot of useful information for you to finish the task.

WHAT ABOUT DISTANCE-BASED JOIN?

A	B	C
a1	b1	1
a2	b2	50
a3	b3	100

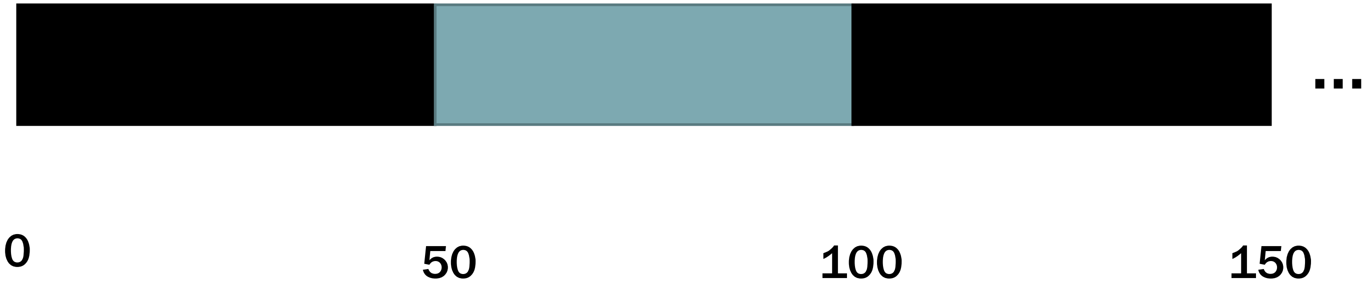
F	D	E
2	d1	e1
99	d2	e2
49	d3	e3
…	…	…
…	…	…

Join Column C of Table 1 and Column F of Table 2 based on the condition that |C-F|<50. Assume that the values of Column C and Column F are positive integers.

If we directly use the values of Column C and Column F, qualified tuples will not be joined together. What should we do?

THE FIRST IDEA

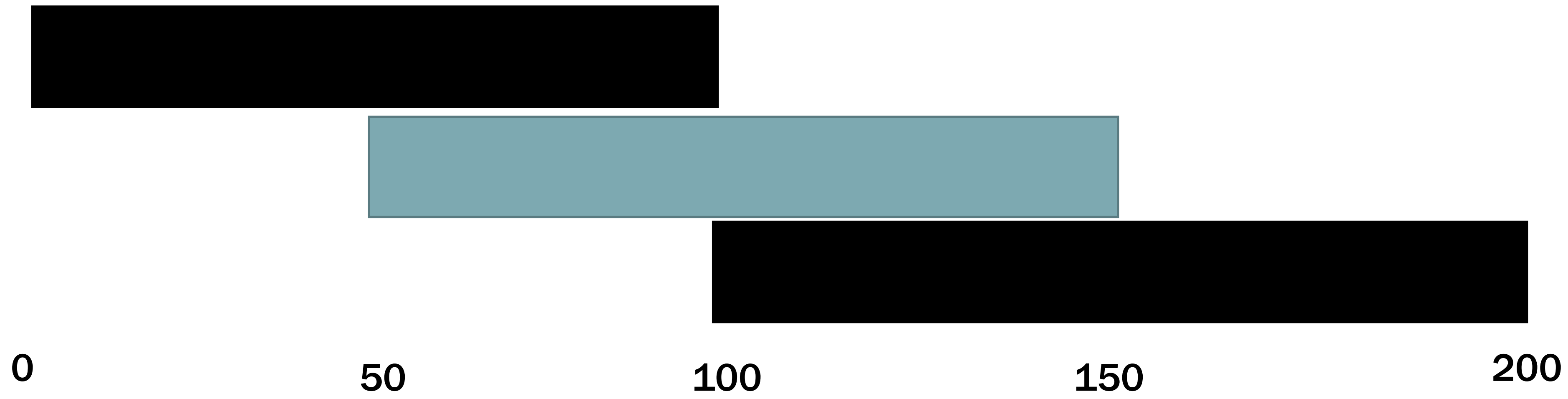
Problem solved?



			Key
Bucket 1	[0, 49]	⟶	1
Bucket 2	[50, 99]	⟶	2
Bucket 3	[100, 149]	⟶	3
Bucket 4	[150, 199]	⟶	4
Bucket 5	[200, 249]	⟶	5

Aggregate the values in the same bucket

FIXED THE IDEA



			Key
Bucket 1	[0, 99]	→	1
Bucket 2	[50, 149]	→	2
Bucket 3	[100, 199]	→	3
Bucket 4	[150, 249]	→	4
Bucket 5	[200, 299]	→	5

Any two values whose difference is at most 50 should fall into at least one buckets

INPUT

Assume the input key-value pairs are the following (corresponding to the example):

1 a1;b1;1

2 a2;b2;50

3 a3;b3;100

4 2;d1;e1

5 99;d2;e2

6 49;d3;e3

JOB1: MAP (PSEUDOCODE)

```
Map(int key, String value){
    if (key<table1_size){
        int value_C=get_value_C(value);
        Emit-Intermediate(floor(value_C/50), “T1:”+value);
        Emit-Intermediate(floor(value_C/50)+1, “T1:”+value);
    }
    else{
        int value_F=get_value_F(value);
        Emit-Intermediate(floor(value_C/50), “T2:”+value);
        Emit-Intermediate(floor(value_C/50)+1, “T2:”+value);
    }
}
```

JOB1: REDUCE (PSEUDOCODE)

```
Reduce(int key, iterator<String> values){
    List value_C_list, value_F_list, value_T1_list, value_T2_list;
    for(String value : values){
      if(value starts with “T1”){
          int value_C=get_value_C(value);
          value_C_list.add(value_C);
          value_T1_list.add(get_tuple(value));
      } else{
          int value_F=get_value_F(value);
          value_F_list.add(value_F);
          value_T2_list.add(get_tuple(value));
      }
    }
```

```
for(int i=0;i<value_C_list.size();i++)
    for(int j=0;j<value_F_list.size();j++){
        int value_C=value_C_list[i];
        int value_F=value_F_list[j];
        if(|value_C-value_F|<50){
            Emit(“”, value_T1_list[i]+value_T2_list[j]);
        }
    }
}
```

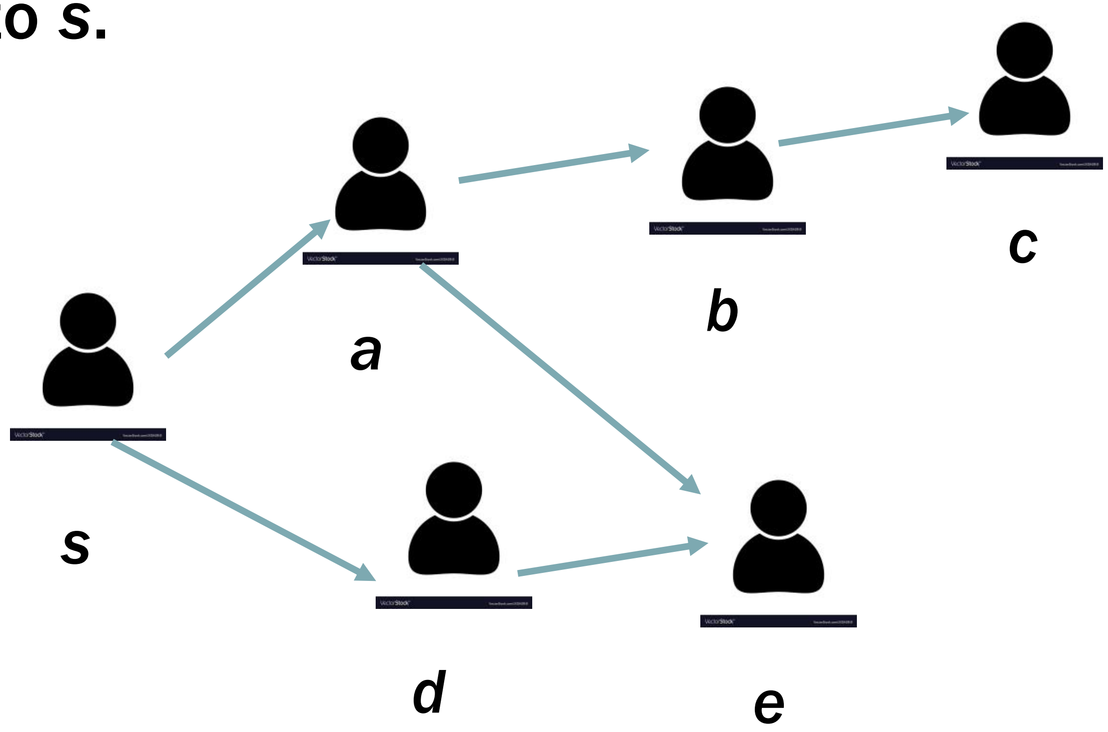
JOB2: REMOVE DUPLICATES

The output of Job1 may contain duplicates (why?)

It is easy to remove duplicates by another MapReduce job.

EXAMPLE: SHORTEST PATH COMPUTATION

Given a directed social network in the following form, find all the people within k-hops from a given source node *s*, and the corresponding hop distance to *s*.



EXAMPLE: SHORTEST PATH COMPUTATION

Given a directed social network in the following form, find all the people within k-hops from a given source node *s*, and the corresponding hop distance to *s*.

Example results (k=2):

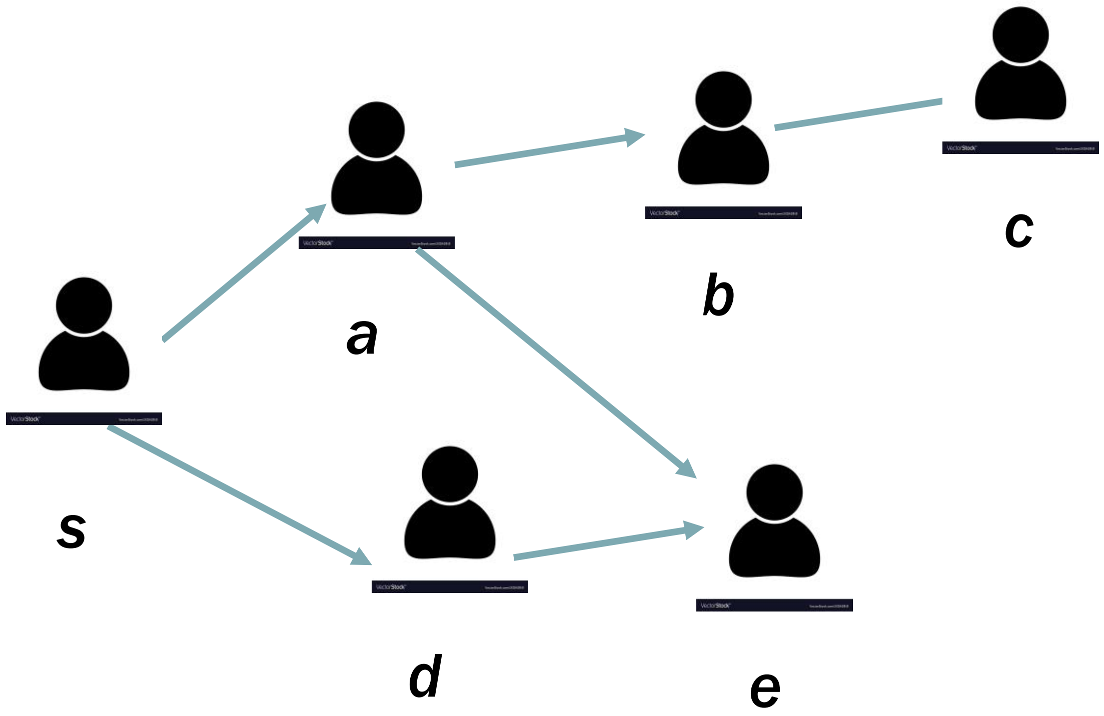
a,d,e,b

dis(s,a)=1

dis(s,d)=1

dis(s,e)=2

dis(s,b)=2



EXAMPLE: SHORTEST PATH COMPUTATION

Single-Machine Algorithm: Dijkstra's algorithm

- An efficient implementation of Dijkstra's algorithm often uses priorityqueue

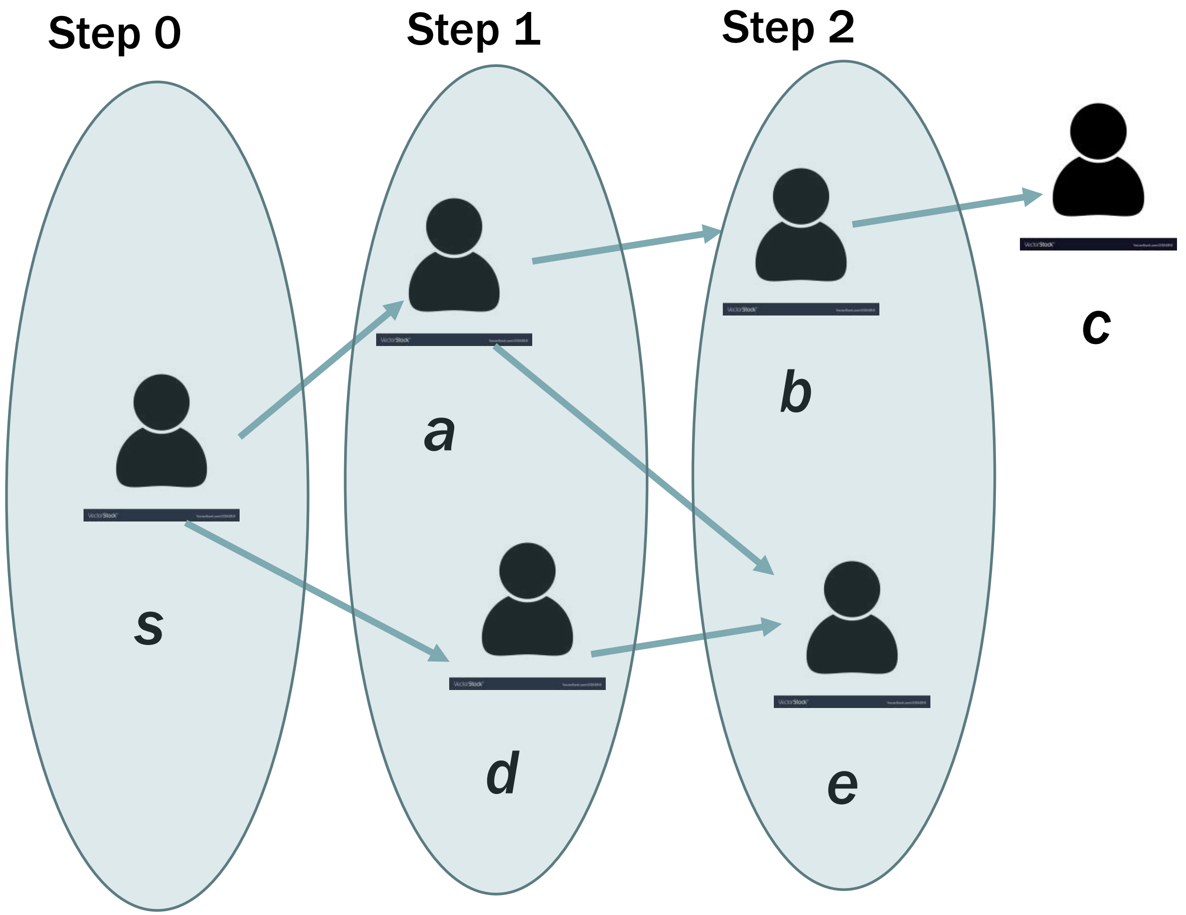
Not easy to parallelize the computation in MapReduce

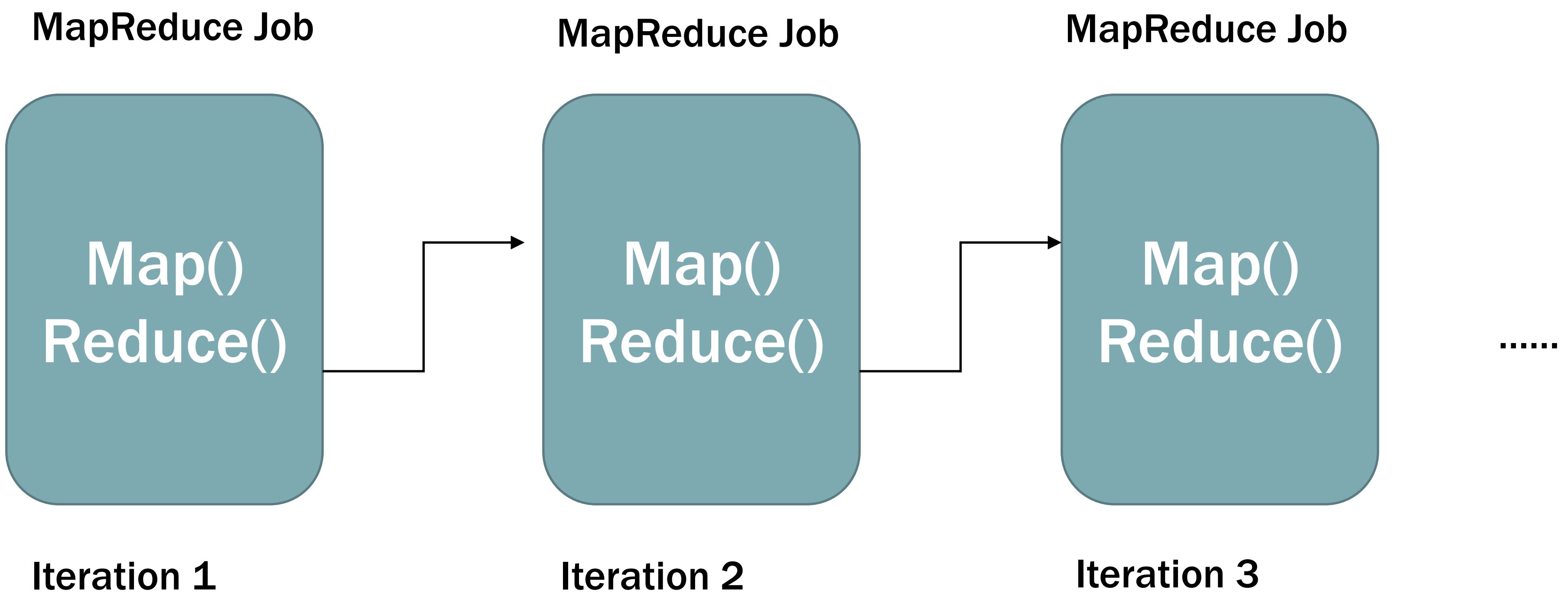
IDEA FOR DESIGNING MAPREDUCE FUNCTIONS

- Solution to the problem can be defined inductively

Intuition

- Initial step (Step 0): dis(s)=0   dis(v)=$+\infty$ for any other v.
- Step 1: For any node v_1 that is 1-hop from s, set $d(v_1)=\min\{d(v_1), 1\}$
- Step 2: For any node v_2 that can be reached in 2-hops from s, set $d(v_2)=\min\{d(v_2), 2\}$
- …
- Step K : For any node v_K that can be reached K-hops from s, set $d(v_K)=\min\{d(v_K),K\}$


Step 0
Step 1
Step 2
s
a
b
c
d
e

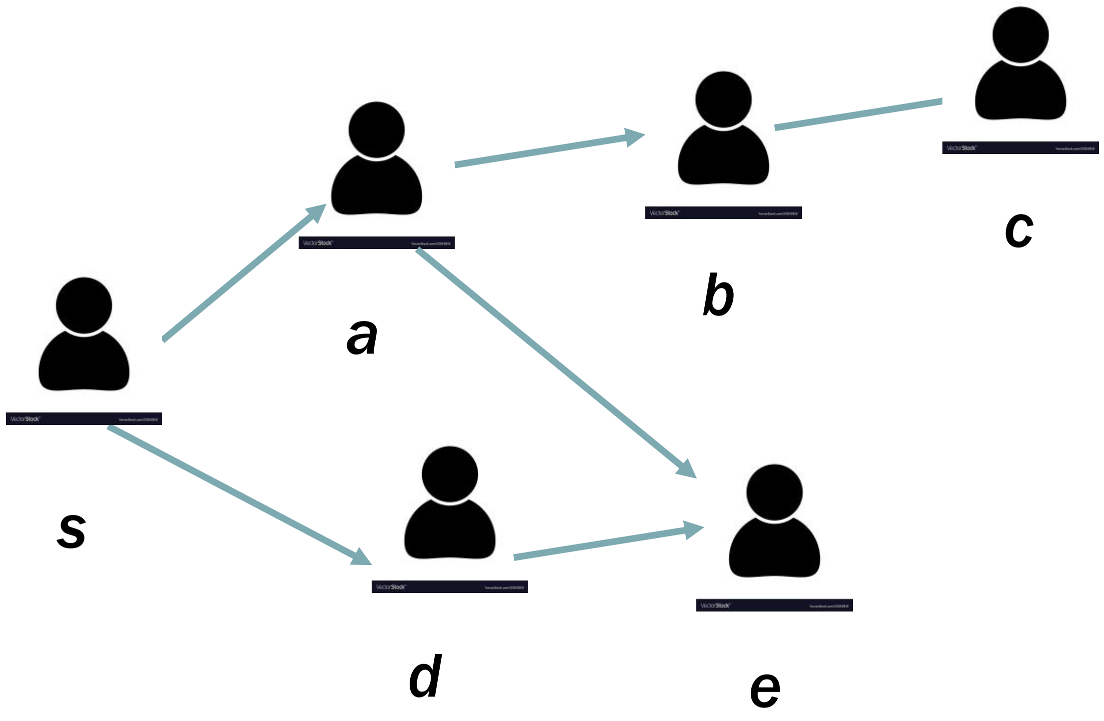

MapReduce Job
MapReduce Job
MapReduce Job
Map()
Reduce()
Map()
Reduce()
Map()
Reduce()
......
Iteration 1
Iteration 2
Iteration 3

INPUT KEY-VALUE PAIRS

Data representation:

Each key-value pair stores the information of an edge

- ❑ Key: nodeID
- ❑ Value: nodeID
- ❑ Example:
  - (s, a)
  - (s, d)
  - (a, e)
  - (d, e)
  - (a, b)
  - (b, c)



Each MapReduce iteration advances the “frontier” by one hop

- Subsequent iterations include more and more reachable nodes as frontier expands
- Multiple iterations are needed to explore the nodes reachable in K hops

How to preserve the graph structure?

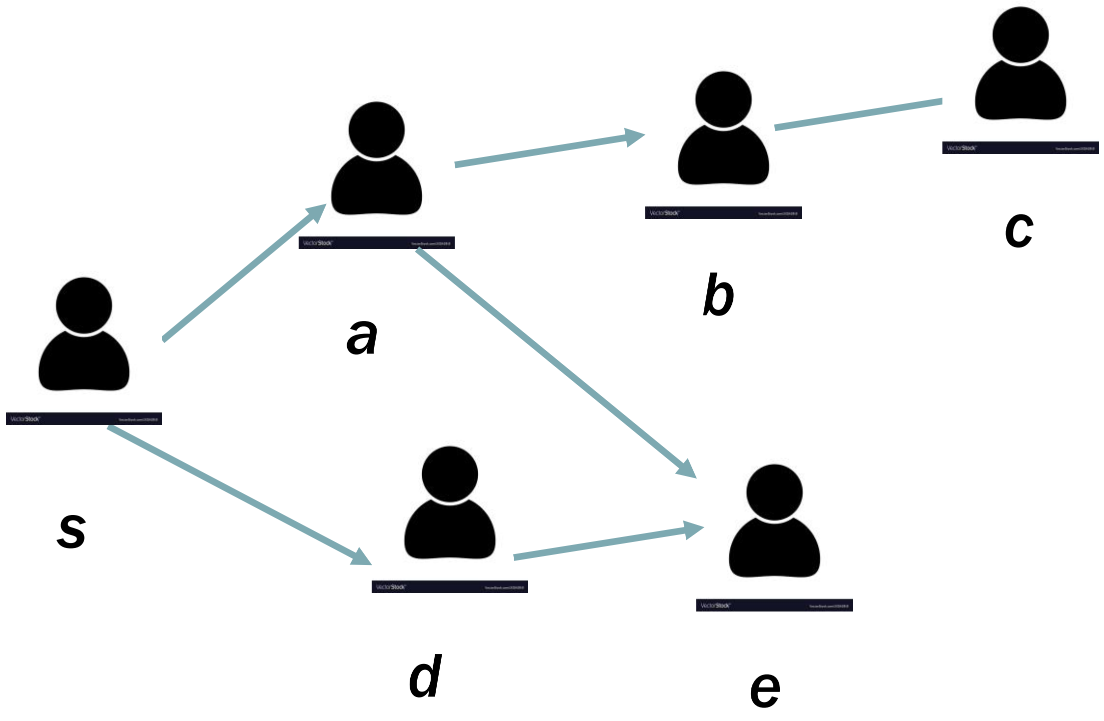
- Solution: map function emits (node, neighbor list) as well

FIRST MAPREDUCE JOB

Purpose:

Each key-value pair stores the information of an edge

- Key: nodeID
- Value: nodeID
- Example:
- (s, a)
- (s, d)
- (a, e)
- (d, e)
- (a, b)
- (b, c)



JOB0(STEP0):MAP

```
map(String nodeid_from, String nodeid_to) {
{
      Emit-Intermediate(nodeid_from, nodeid_to);
}
```

JOB0(STEP0):REDUCE

```
reduce(String nodeid, Iterator<String> neighbors) {
        if(nodeid.equals("s")){
            Emit("s", "D:0");//"D" indicates distance
        }

        Emit(nodeid, "N:"+ToString(neighbors));//"N" indicates
neighbors
}
```

Note: ToString() makes the list of neighbors as a string

EXAMPLE FOR JOB0

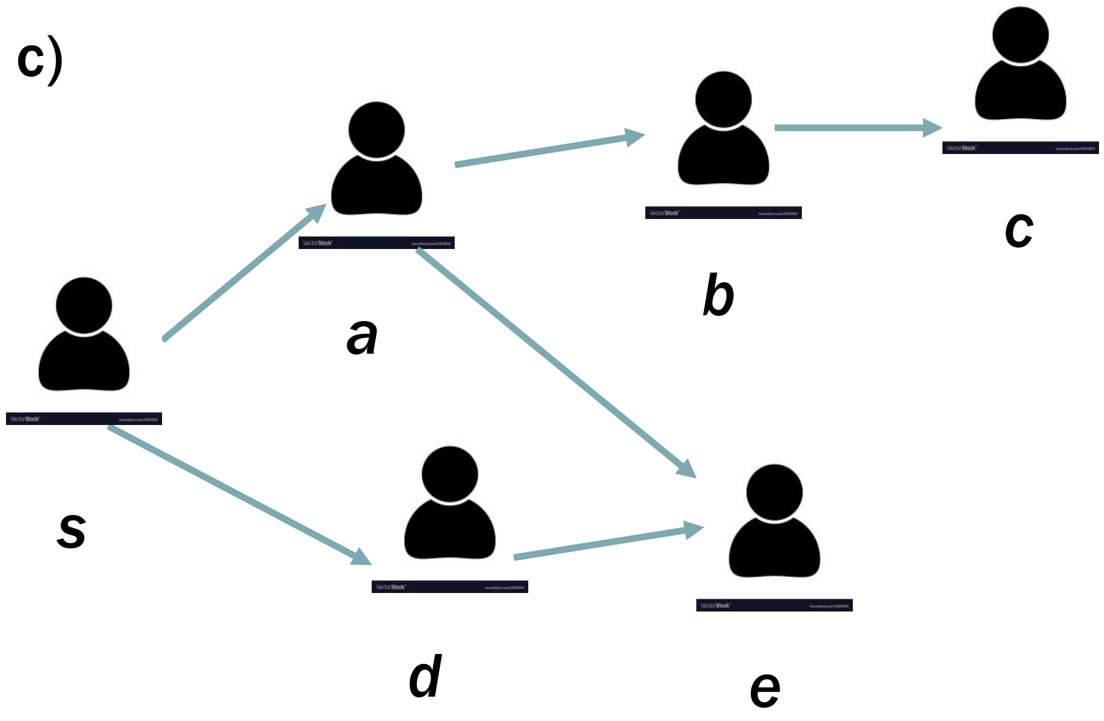
Job0:

Input

- (s, a) (s, d) (a, e) (d, e) (a, b) (b, c)

Output

- (s,"D:0")
- (s, “N:a;d”)
- (a, “N:b;e”)
- (d, “N:e”)
- (b, “N:c”)



SUBSEQUENT JOBS (STEPS 1-K): MAP

```
map(String nodeid, String value) {
{
        Emit-Intermediate(nodeid, value);
}
```

SUBSEQUENT JOBS (STEPS 1-K): REDUCE

```
reduce(String nodeid, Iterator<String> values) {
    d_min=+∞;
    Iterator<String> neighbors;
    for( value in values ){
        if (value starts with “N”){
            neighbors= ToNeighbors(value);
            Emit(nodeid, value);//always send neighbors out
        }
        else{
            if( ToInteger(value)<d_min)
                d_min=ToInteger(value);
        }
    }
    if(d_min!=+∞){
        for (neighbor in neighbors){
            Emit(neighbor, ToString(“D:”,d_min+1));
            // possible distances for neighbors
        }
        Emit(nodeid, ToString(“D:”,d_min));
    }
}
```

EXAMPLE FOR JOB1

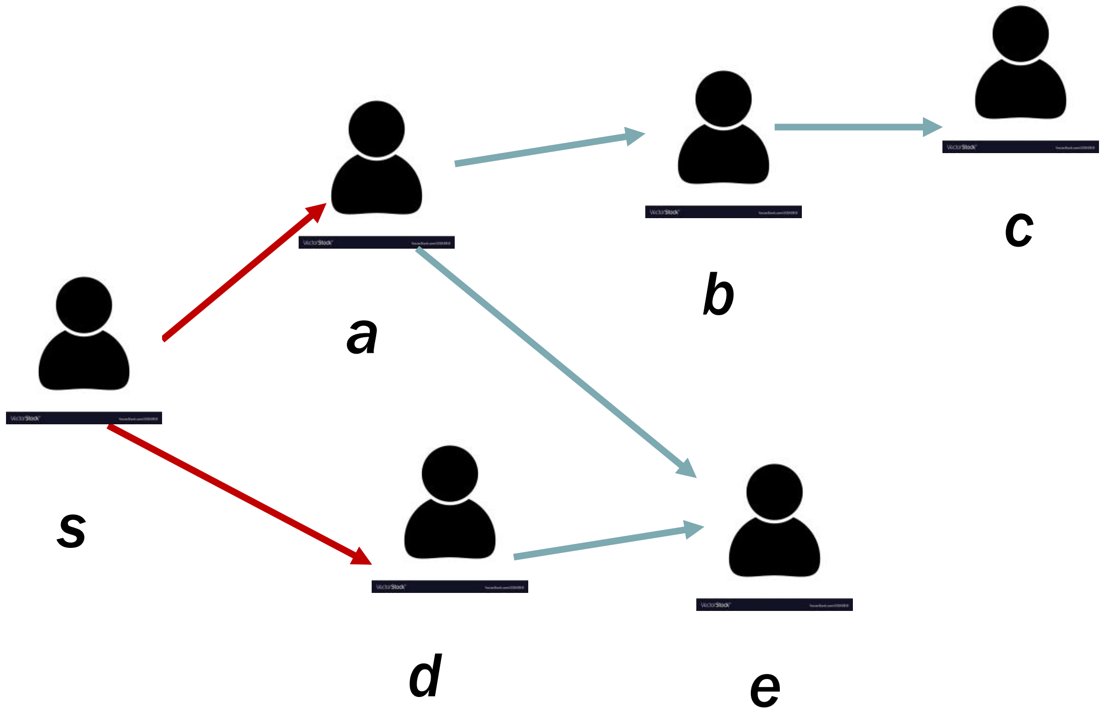
Job1:

Input

- (s,”D:0”) (s, “N:a;d”) (a, “N:b;e”) (d, “N:e”) (b, “N:c”)

Output

- (s,”D:0”)
- (s, “N:a;d”)
- (a, “N:b;e”)
- (d, “N:e”)
- (b, “N:c”)
- (a, “D:1”)
- (d, “D:1”)



EXAMPLE FOR JOB2

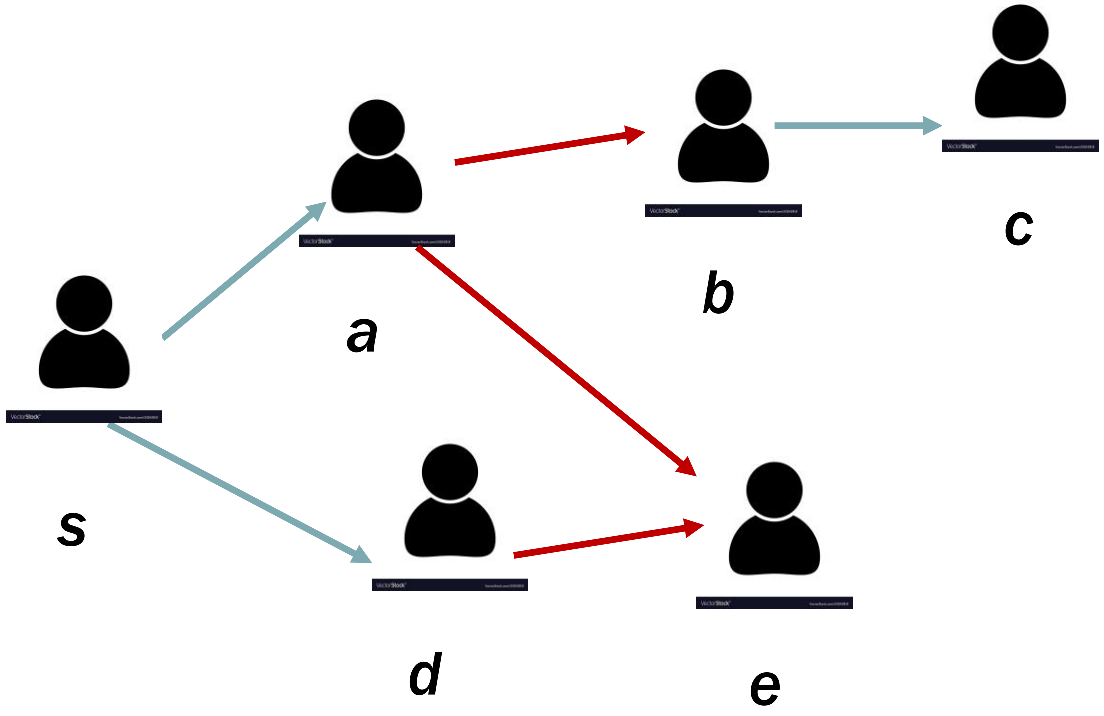
Job2:

Input

- (s,”D:0”) (s, “N:a;d”) (a, “N:b;e”) (d, “N:e”) (b, “N:c”) (a, “D:1”) (b, “D:2”)

Output

- (s,”D:0”)
- (s, “N:a;d”)
- (a, “N:b;e”)
- (d, “N:e”)
- (b, “N:c”)
- (a, “D:1”)
- (d, “D:1”)
- (b, “D:2”)
- (e, “D:2”)
- (e, “D:2”)



LAST JOB: MAP

```
map(String nodeid, String value) {
{
        Emit-Intermediate(nodeid, value);
}
```

LAST JOB: REDUCE

```
reduce(String nodeid, Iterator<String> values) {
    d_min=+∞;
    for( value in values ){
        if (value starts with "D"){
                if( ToInteger(value)<d_min)
                    d_min=ToInteger(value);
        }
    }
    if(d_min!=+∞)
        Emit(nodeid, ToString("D:",d_min));
}
```

EXAMPLE LAST JOB

Job2:

Input

- (s,”D:0”) (s, “N:a;d”) (a, “N:b;e”) (d, “N:e”) (b, “N:c”) (a, “D:1”) (d, “D:1”) (b, “D:2”) (e, “D:2”) (e, “D:2”)

Output

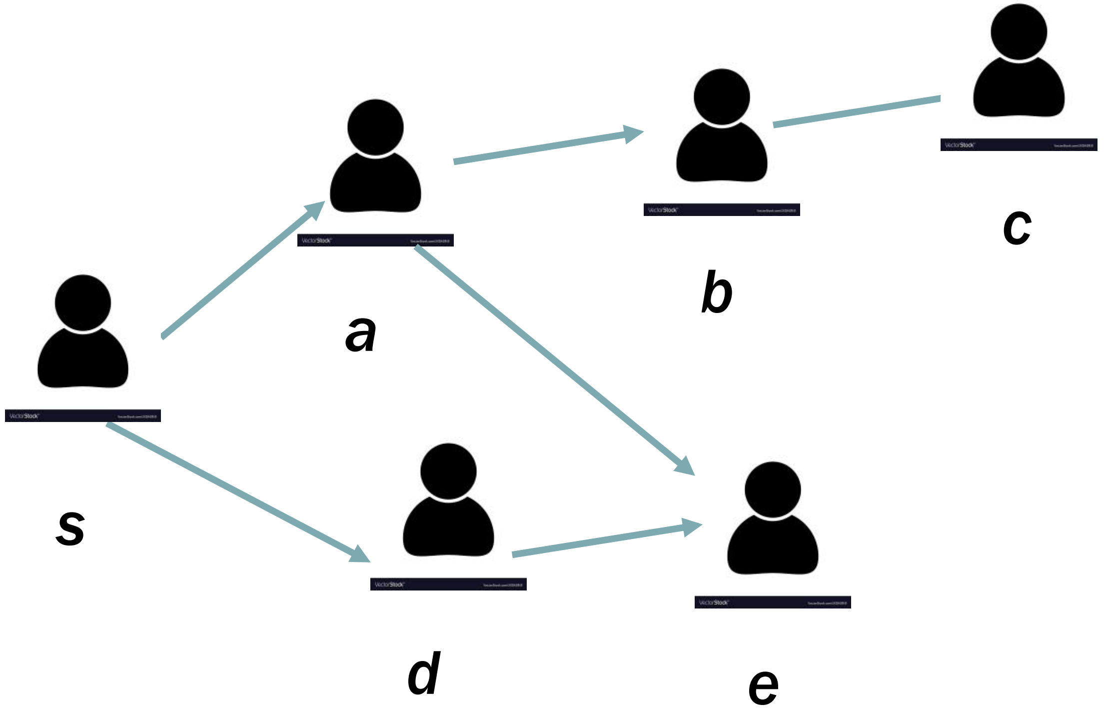
(s,”D:0”)

(a, “D:1”)

(d, “D:1”)

(b, “D:2”)

(e, “D:2”)



OPTIMIZATION?

FURTHER EXTENSION

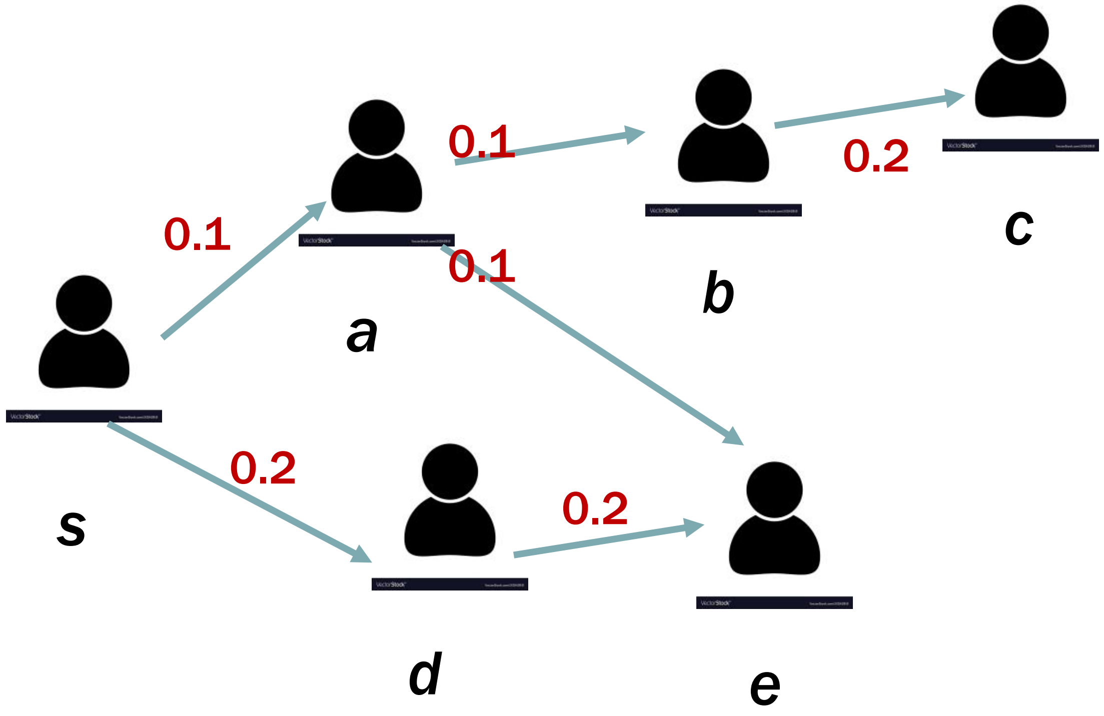
Previously, we considered the *K*-hop shortest path computation in social graphs using MapReduce. Extend the pseudo code to handle weighted graph whose edges may have different weights.

FIRST MAPREDUCE JOB

Purpose:

Each key-value pair stores the information of an edge

- Key: nodeID
- Value: nodeID; weight
- Example:
- (s, “a; 0.1”)
- (s, “d; 0.2”)
- (a, “e; 0.1”)
- (d, “e; 0.2”)
- (a, “b; 0.1”)
- (b, “c; 0.2”)



JOB0(STEP0):MAP

```
map(String nodeid_from, String nodeid_to_and_weight) {
{
    Emit-Intermediate(nodeid_from, nodeid_to_and_weight);
}
```

JOB0(STEP0):REDUCE

```
reduce(String nodeid, Iterator<String> neighbors) {
        if(nodeid.equals("s")){
            Emit("s", "D:0");//"D" indicates distance
        }

        Emit(nodeid, "N:"+ToString(neighbors));//"N" indicates neighbors
}
```

Note: ToString() makes the list of neighbors as a string

EXAMPLE FOR JOB0

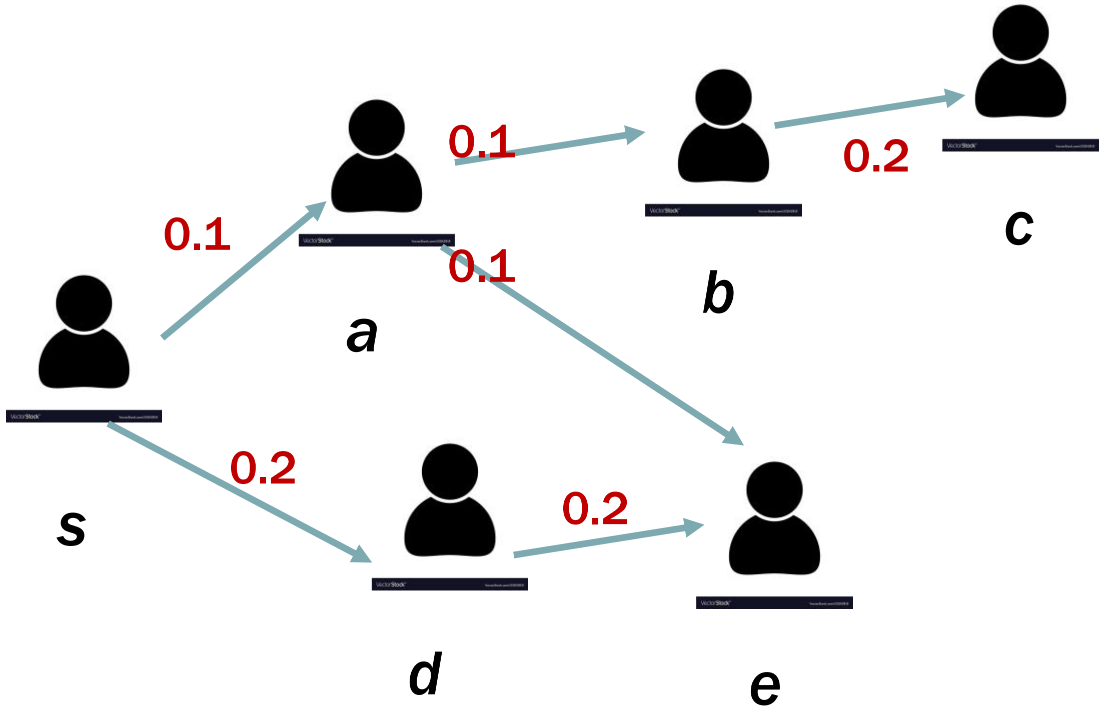
Job0:

Input

- (s, a;0.1) (s, d;0.2) (a, e;0.1) (d, e;0.2) (a, b;0.1) (b, c;0.2)

Output

- (s,”D:0”)
- (s, “N:a;0.1;d;0.2”)
- (a, “N:b;0.1;e;0.1”)
- (d, “N:e;0.2”)
- (b, “N:c;0.2”)



SUBSEQUENT JOBS (STEPS 1-K): MAP

```
map(String nodeid, String value) {
{
        Emit-Intermediate(nodeid, value);
}
```

SUBSEQUENT JOBS (STEPS 1-K): REDUCE

```
reduce(String nodeid, Iterator<String> values) {
    d_min=+∞;
    Iterator<String> neighbors;
    Iterator<Double> weights;
    for( value in values ){
        if (value starts with “N”){
            neighbors= ToNeighborsNodes(value);
            weights= ToNeighborsWeights(value);
            Emit(nodeid, value);
        }else{
            if( GetDoubleValue(value)<d_min)
                d_min=GetDoubleValue(value);// GetDoubleValue removes “N” and convert the value to double.
        }
    }
    if(d_min!=+∞){
        for (int u=0;u<neighbors.size();u++){
            Emit(neighbors[i], ToString(“D:”,d_min+weights[i]));
        }
        Emit(nodeid, ToString(“D:”,d_min));
    }
}
```

LAST JOB: MAP

```
map(String nodeid, String value) {
{
        Emit-Intermediate(nodeid, value);
}
```

LAST JOB: REDUCE

```
reduce(String nodeid, Iterator<String> values) {
    d_min=+∞;
    for( value in values ){
        if (value starts with "D"){
                if(GetDoubleValue(value)<d_min)
                        d_min=GetDoubleValue(value);
        }
    }
    if(d_min!=+∞)
        Emit(nodeid, ToString("D:",d_min));
}
```

EXAMPLE: PAGERANK

- Google’s famous algorithms for ranking webpages.
- A measure of the “importance/quality” of a page.
- Widely adopted for ranking search results
- Intuition
  - Consider a random surfer that starts at a random webpage
  - He follows out-going links in a random manner with probability $(1-\alpha)$
    He jumps to a random webpage with probability α
  - PageRank is the probability that the random surfer will arrive at a given webpage

MORE FORMALLY

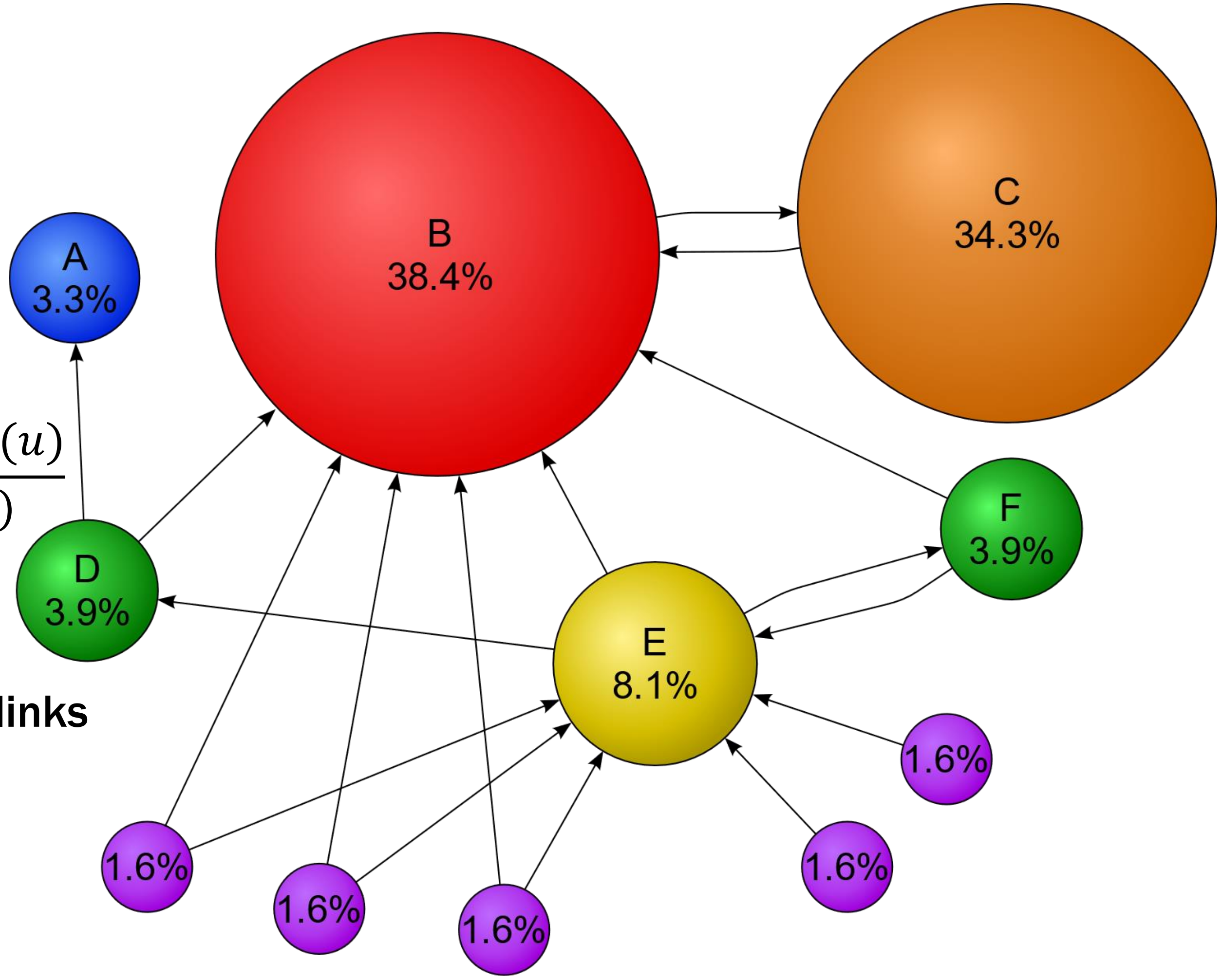
PageRank of a page is based on the PageRank of the pages which link to it

a value in (0, 1)

$$\mathrm{Page}Rank(v) = \frac{\alpha}{n} + (1-\alpha) \times \sum_{u\ links\ to\ v} \frac{PageRank(u)}{d\_out(u)}$$

Number of web pages

Number of out-going links



PAGERANK COMPUTATION

Step 1: Initialize PageRank(v)=1/n for every webpage.

Step 2: Iteratively apply the formula until convergence

For each webpage *v*, compute

Value of the previous iteration

$$PageRank(v) = \frac{\alpha}{n} + (1-\alpha) \times \sum_{u\ links\ to\ v} \frac{PageRank(u)}{d\_out(u)}$$

MAPREDUCE DESIGNS

Each iteration is a MapReduce job:

Question:

Which features should be aggregated for a page?

$$\text{PageRank}(v) = \frac{\alpha}{n} + (1-\alpha) \times \sum_{u\ links\ to\ v} \boxed{\frac{PageRank(u)}{d\_out(u)}}$$

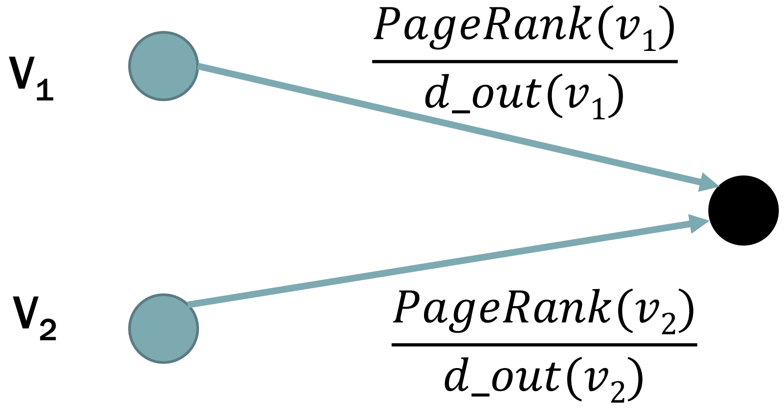
MAPREDUCE DESIGNS

Each iteration is a MapReduce job:

Map:

Input with the current PageRank value of a webpage *v* (or set to PageRank(v)=1/n in the 1st job).

For each out-neighbor link (*u*) of current webpage *v* (namely, *v*→*u*), compute $\frac{PageRank(v)}{d\_out(v)}$ and Emit-Intermediate(*u*,$\frac{PageRank(v)}{d\_out(v)}$)



Reduce:

Each webpage *u* receives the PageRank portion (stored in values of reduce()) from its incoming links, and hence it can compute

$$\sum_{u\ links\ to\ v} \frac{PageRank(u)}{d\_out(u)}$$

Use the formula to update PageRanks for the next job

MAP

```
Map(String nodeid, String value_and_neighbors){
// value stores neighbor list and PageRank of nodeid
    double PR=getPageRank(value_and_neighbors);
    List<String> neighbors = getNeighbors(value_and_neighbors);
    for(each neighbor in neighbors){
        Emit-Intermediate(neighbor, “P:”+PR/neighbors.size());
    }
    Emit-Intermediate(nodeid, “N:”+ToString(neighbors));
}
```

REDUCE

```
Reduce(String nodeid, Iterator values){
    double PR=0.0;
    String neighbors;
    for(value in values){
      if(value starts with “P”){
          PR = PR + getPageRank(value) × (1 − α);
      }
      else{
          neighbors = getNeighbors(value);
      }
    }
    PR=PR+ α/n ;
    Emit(nodeid, PR+”;”+ neighbors);
}
```

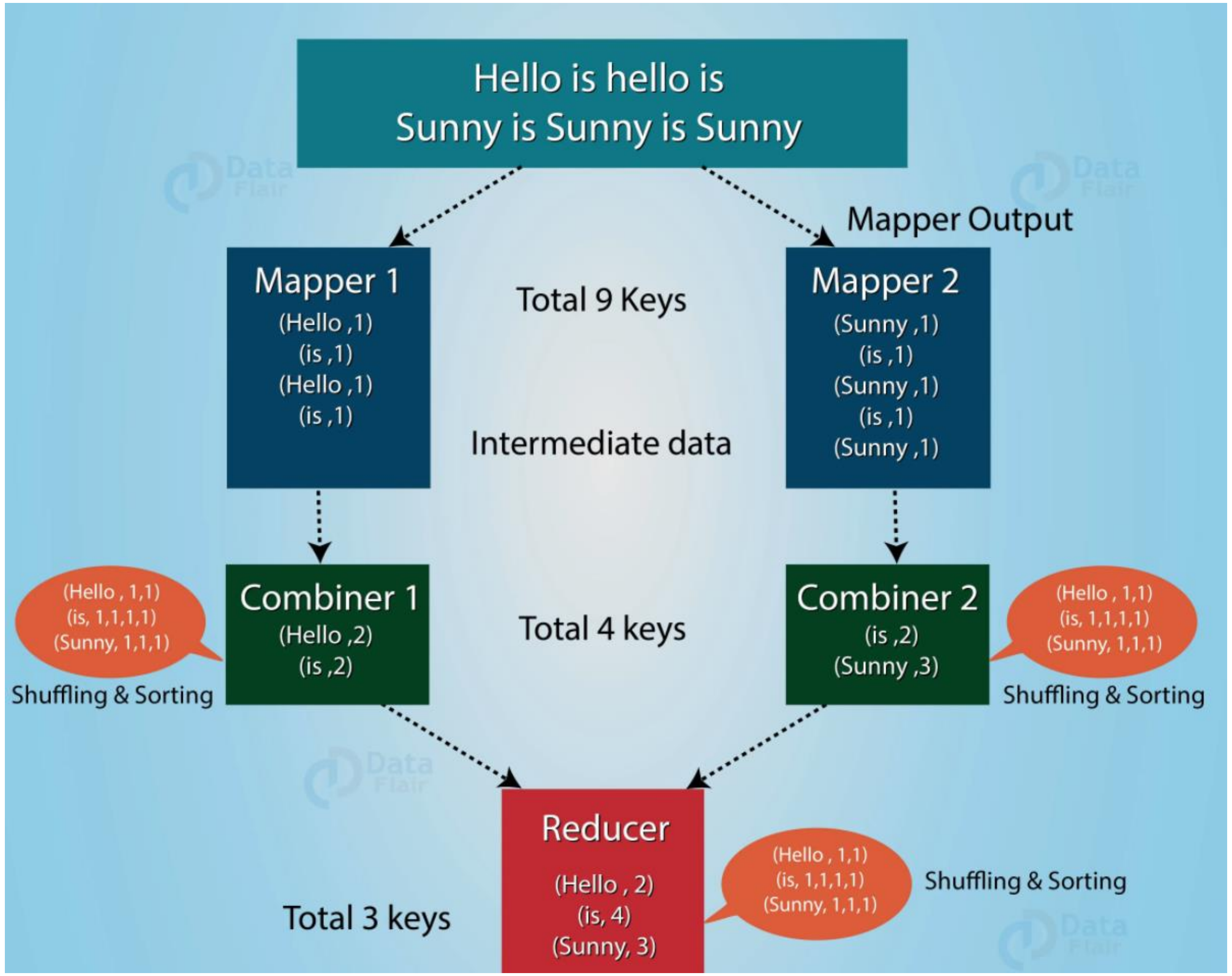
SUMMARY AND OPEN DISCUSSIONS

MapReduce model is very powerful.

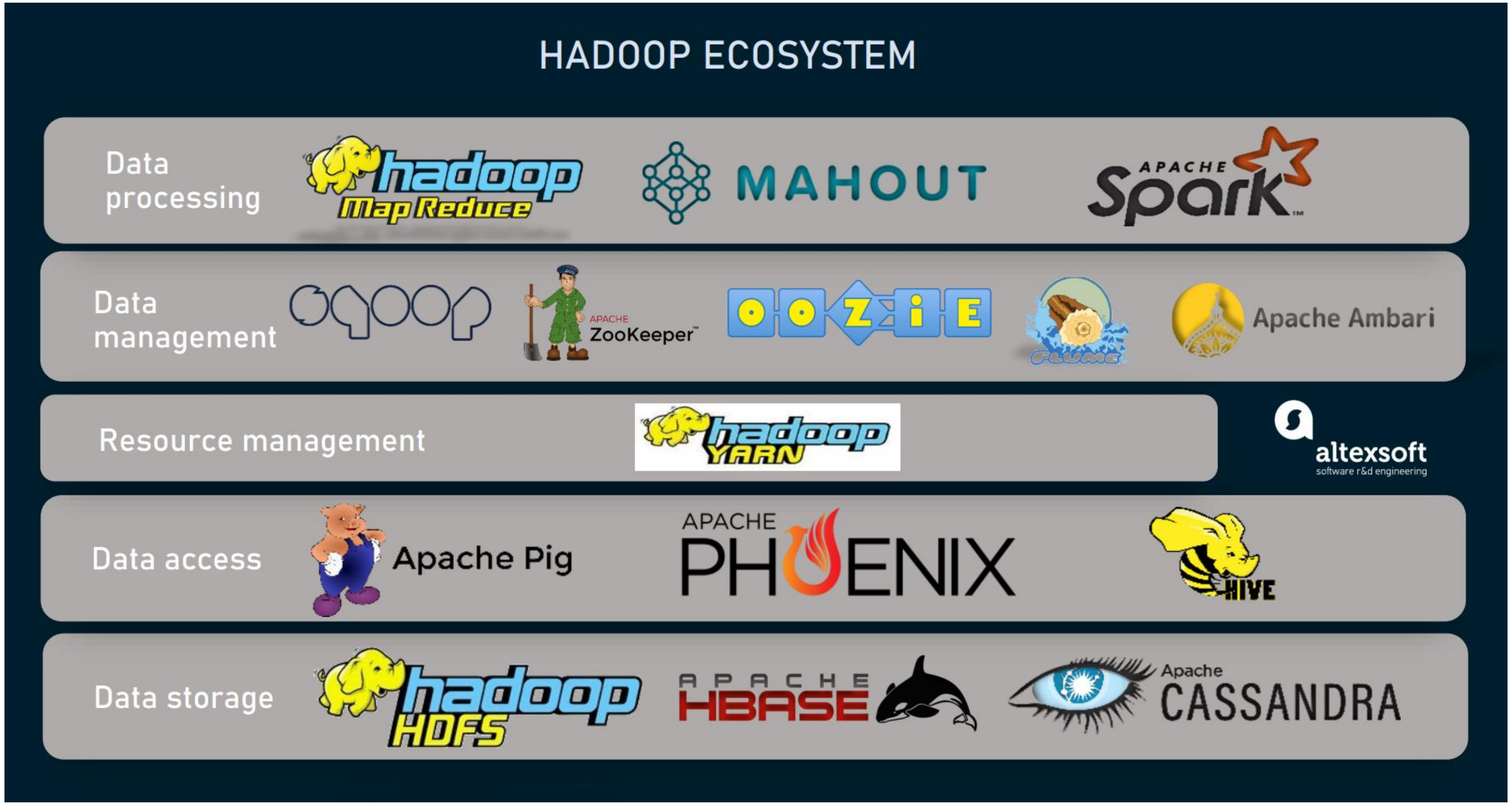
- ❑ Database operations (join)
- ❑ Graph based queries (shortest paths, PageRanks)
- ❑ …

1. Any more examples?
2. Possible optimization

COMBINER


Hello is hello is
Sunny is Sunny is Sunny
Mapper Output
Mapper 1
(Hello ,1)
(is ,1)
(Hello ,1)
(is ,1)
Total 9 Keys
Intermediate data
Mapper 2
(Sunny ,1)
(is ,1)
(Sunny ,1)
(is ,1)
(Sunny ,1)
(Hello , 1,1)
(is, 1,1,1,1)
(Sunny, 1,1,1)
Combiner 1
(Hello ,2)
(is ,2)
Shuffling & Sorting
Total 4 keys
Combiner 2
(is ,2)
(Sunny ,3)
(Hello , 1,1)
(is, 1,1,1,1)
(Sunny, 1,1,1)
Shuffling & Sorting
Reducer
(Hello , 2)
(is, 4)
(Sunny, 3)
Total 3 keys
(Hello , 1,1)
(is, 1,1,1,1)
(Sunny, 1,1,1)
Shuffling & Sorting

HADOOP → SPARK



Source: https://www.altexsoft.com/blog/hadoop-vs-spark/

We finish lectures for Distributed Systems!

Next lecture: NoSQL and Key-Value Stores