15-319 / 15-619 Cloud Computing

Recitation 13

April 18th 2017

Overview

- **Last week's reflection**
  - Team project phase 1
  - Quiz 11
- **This week's schedule**
  - Project 4.2
- **Twitter Analytics: The Team Project**

Conceptual Modules to Read on OLI

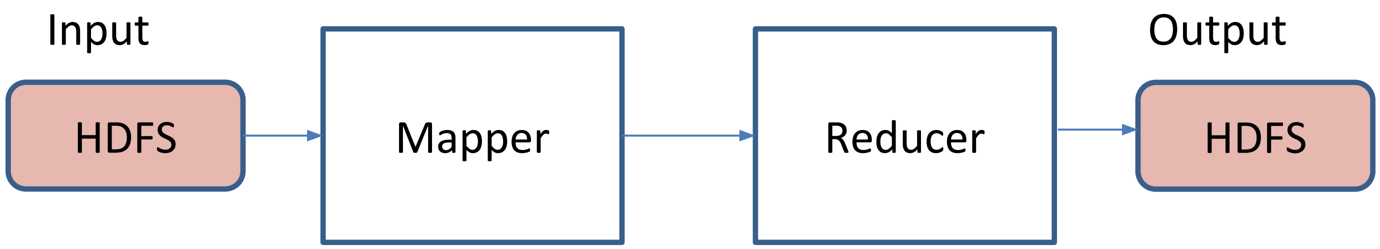
- UNIT 5: Distributed Programming and Analytics Engines for the Cloud
  - Module 18: Introduction to Distributed Programming for the Cloud
  - Module 19: Distributed Analytics Engines for the Cloud: MapReduce
  - Module 20: Distributed Analytics Engines for the Cloud: Spark
  - Module 21: Distributed Analytics Engines for the Cloud: GraphLab
  - Module 22: Message Queues and Stream Processing

Project 4

- Project 4.1, Batch Processing with MapReduce
  - MapReduce Programming
- Project 4.2
  - Iterative Batch Processing Using Apache Spark
- Project 4.3
  - Stream Processing using Kafka/Samza

Typical MapReduce Batch Job

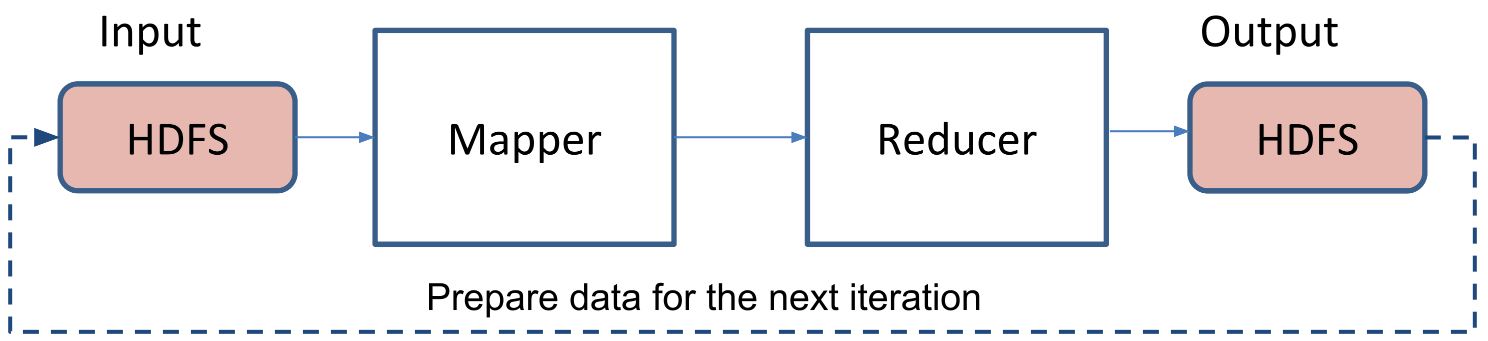
- Simplistic view of a MapReduce job



- You simply write code for the
  - Mapper
  - Reducer
- Inputs are read from disk and outputs are written to disk
  - Intermediate data is spilled to local disk

Iterative MapReduce Jobs

- Some applications require iterative processing
- Eg: Machine Learning, etc.



- MapReduce: Data is always **spilled** to disk
  - This leaded to added overhead for each iteration
  - Can we keep data in memory? Across Iterations?
  - How do you manage this?

Resilient Distributed Datasets (RDDs)

- New abstraction, RDDs
  - can be in-memory or on disk
  - are read-only objects
  - are partitioned across the cluster
    - partitioned across machines based on a range or the hash of a key in each record

Operations on RDDs

- Loading data

```
>>>input_RDD = sc.textFile("text.file")
```

- Transformation
  - Apply an operation and derive a new RDD

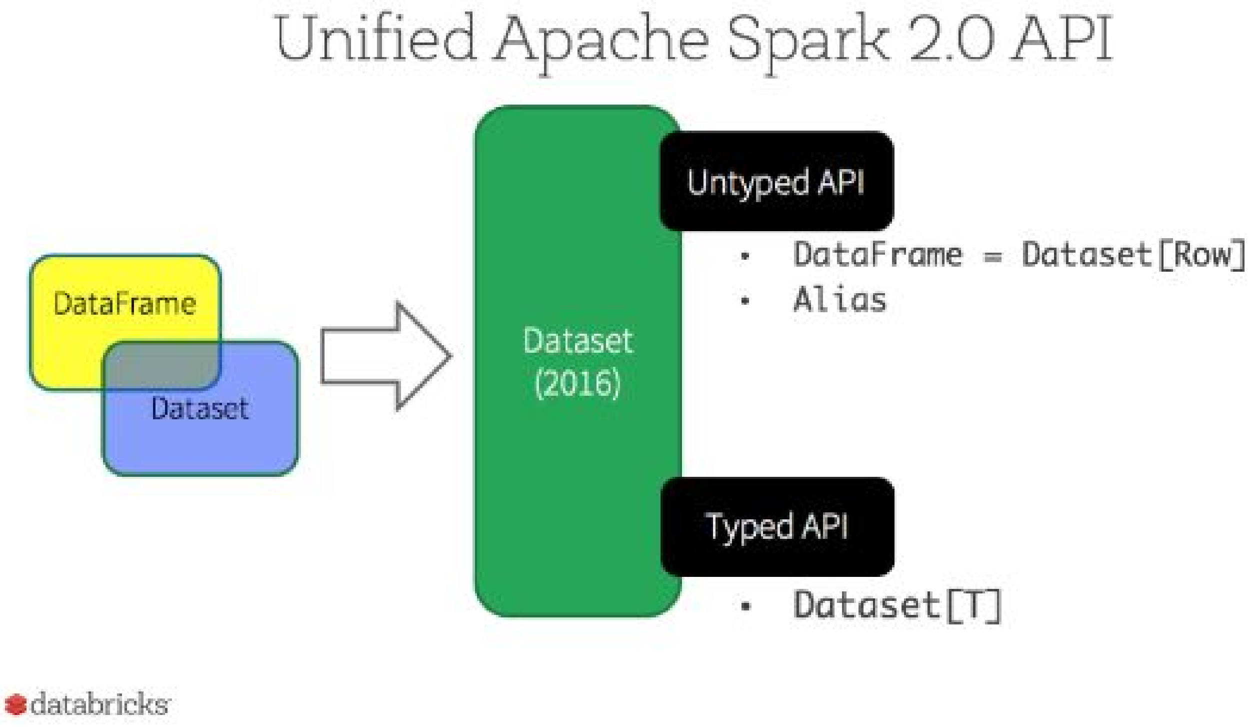
```
>>>transform_RDD =  input_RDD.filter(lambda x: "abcd" in x)
```

- Action
  - Computations on an RDD and return a single object

```
>>>print "Number of “abcd”:" + transform_RDD.count()
```

DataFrames

- Like an RDD, a DataFrame is an immutable distributed collection of data, organized into named columns, like a table in a relational database.



DataFrames

- Json:

```
{"name":"Michael"}{"name":"Andy", "age":30} {"name":"Justin", "age":19}
```

- Load data:

```
val df = spark.read.json("people.json")
```

- Convert dataframe to dataset:

```
case class Person(name: String, age: Long)
val ds = df.as[Person]
```

- SQL:

```
val sqlDF = spark.sql("SELECT name FROM people where age >  20").show()
// +----+
// |name|
// +----+
// |Andy|
// +----+
```

DataFrames

- Json:

```
{"name":"Michael"}{"name":"Andy", "age":30} {"name":"Justin", "age":19}
```

- Load:

```
val df = spark.read.json("people.json")
```

- Convert dataframe to dataset:

```
case class Person(name: String, age: Long)
```

```
val ds = df.as[Person]
```

- SQL:

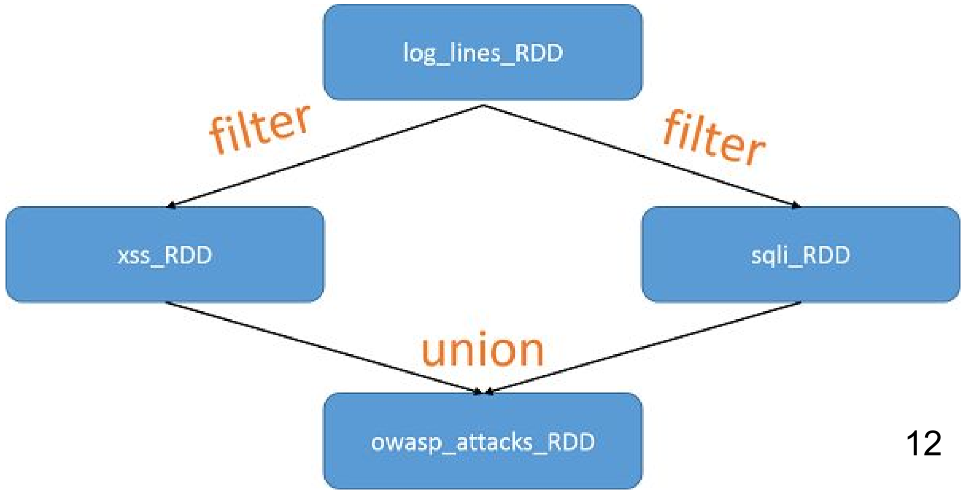
```
val sqlDF = spark.sql("SELECT name FROM people where age >  20").show()
```

- API:

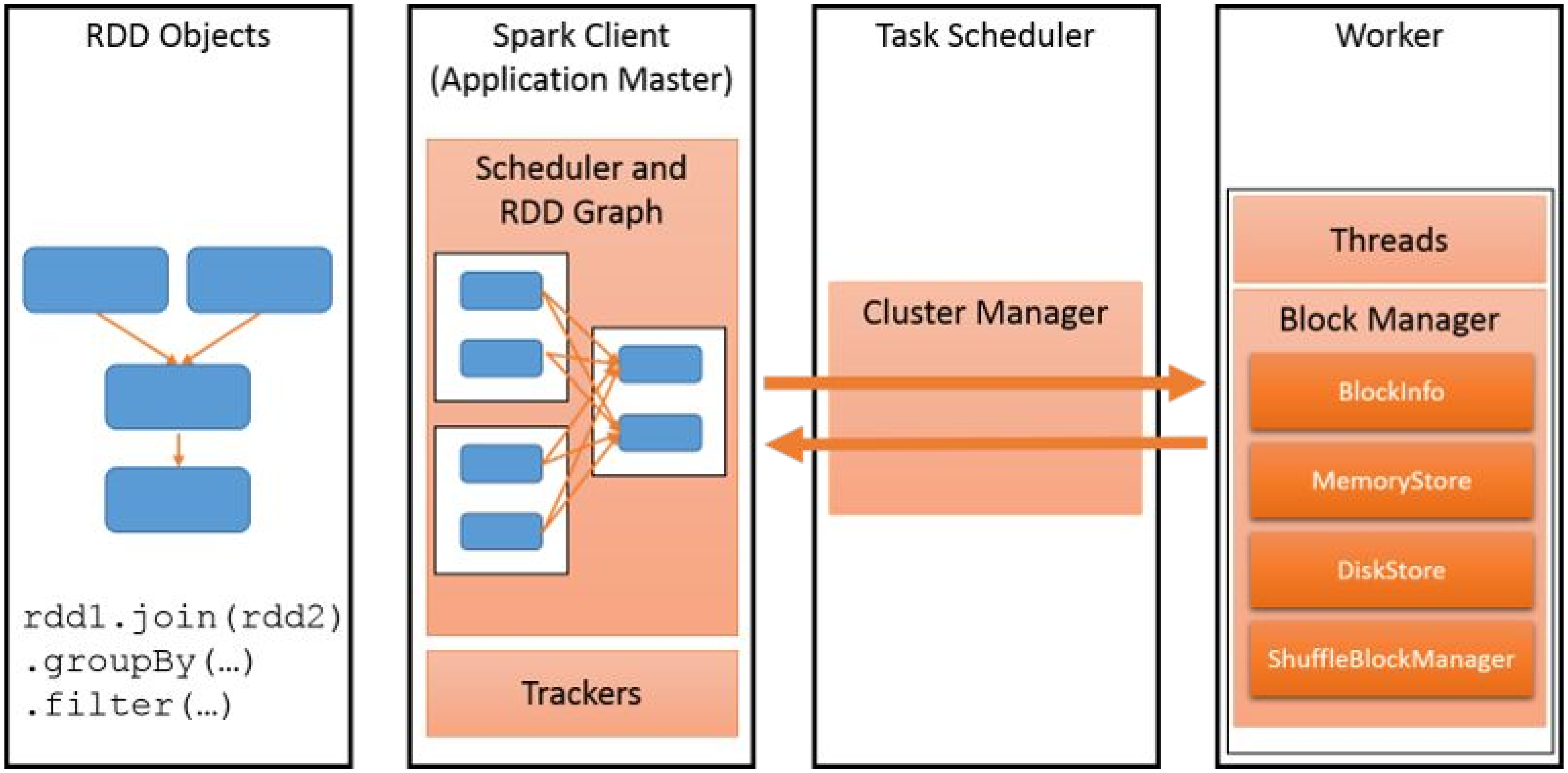
```
df.select("name").filter($"age" > 20).show()
```

RDDs and Fault Tolerance

- Actions create new RDDs
- Instead of replication, recreate RDDs on failure
- Recreate RDDs using lineage
  - RDDs store the transformations required to bring them to current state
  - Provides a form of resilience even though they can be in-memory



The Spark Framework



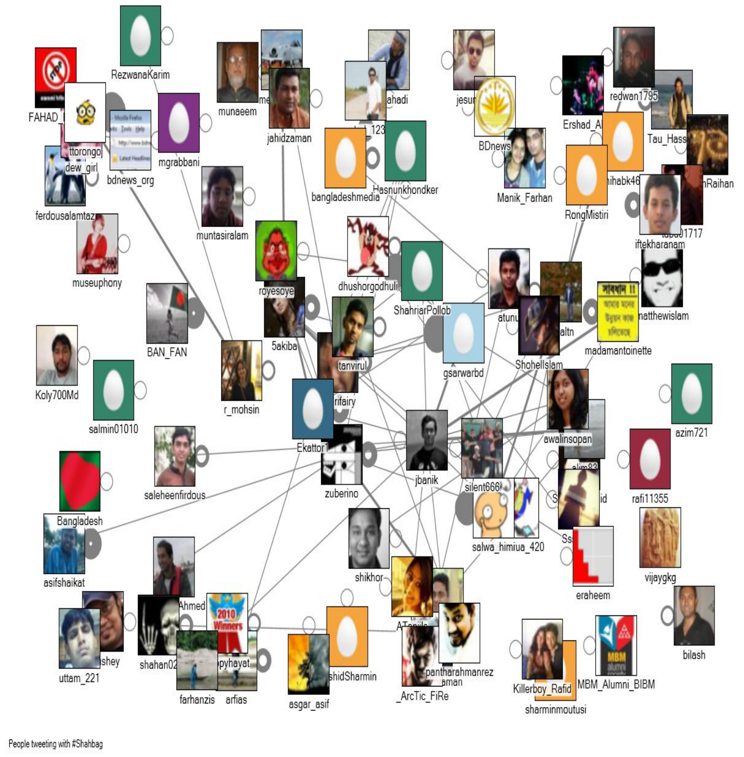
```
rdd1.join(rdd2)
.groupBy(…)
.filter(…)
```

Spark Ecosystem

- Spark SQL
  - Allows running of SQL-like queries against RDDs
- Spark Streaming
  - Run spark jobs against streaming data
- MLlib
  - Machine learning library
- GraphX
  - Graph-parallel framework

Project 4.2

- Use Spark to analyze a Twitter social graph
  - **Task 1**
    - Number of nodes and edges
    - Number of followers for each user
    - RDD vs. Dataframe
  - **Task 2**
    - Run PageRank to compute the influence of users
    - Fast runs get a **bonus**
  - **Task 3**
    - 2nd-degree centrality on the graph using GraphX

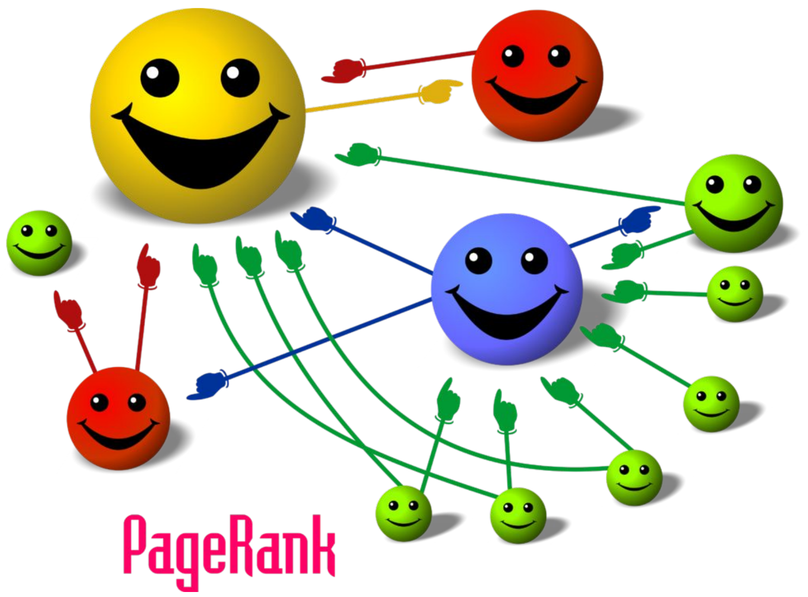


Project 4.2 - Three Tasks

1. Enumerate the Twitter Social Graph
   - Find the number of nodes and edges
   - Edges in the graph are directed. (u, v) and (v, u) should be counted as two edges
   - Find the number of followers for each user
   - Compare **RDD/Dataframe** implementations
2. Rank each user by influence
   - Run PageRank with **10** iterations
   - **All users' scores need to sum up to 1.0 at every iteration.**
3. Second degree centrality
   - Need to use GraphX with Scala.

Task 2: The PageRank Algorithm

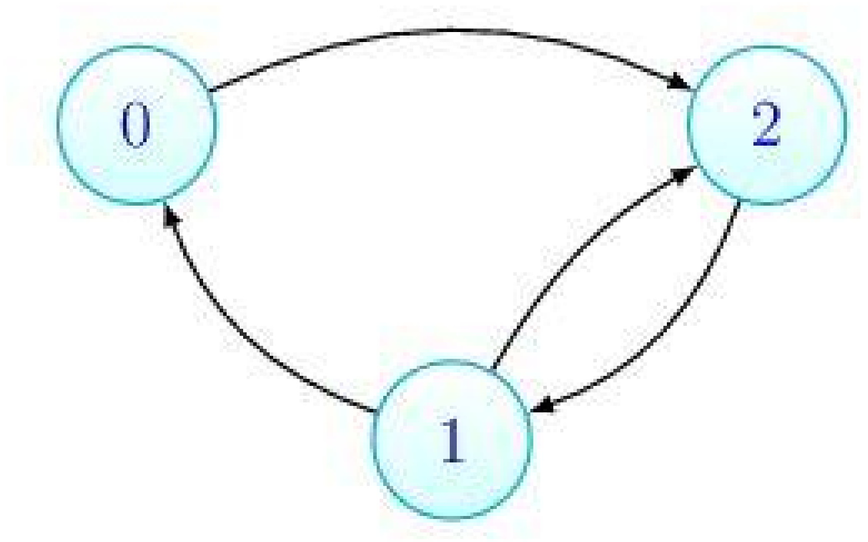
- Give ranks (scores) based on links to them
- A page that has:
  - Links from many nodes ⇒ high rank
  - Link from a high-ranking node ⇒ high rank



The PageRank Algorithm

The PageRank algorithm can find important or influential vertices in a graph.

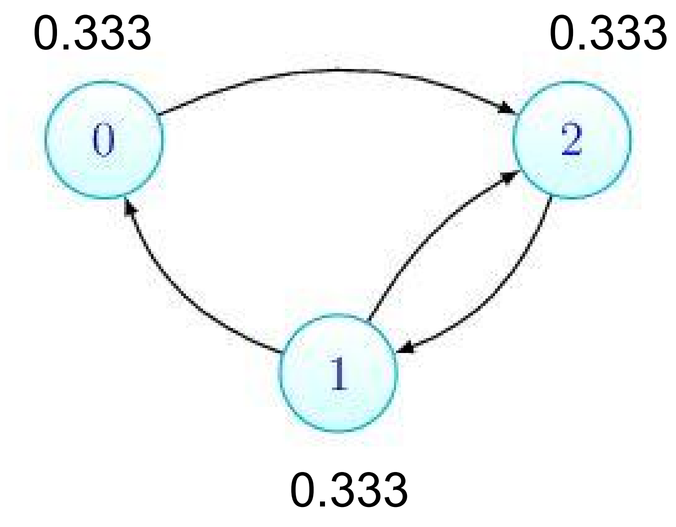
Which node is more important or influential?



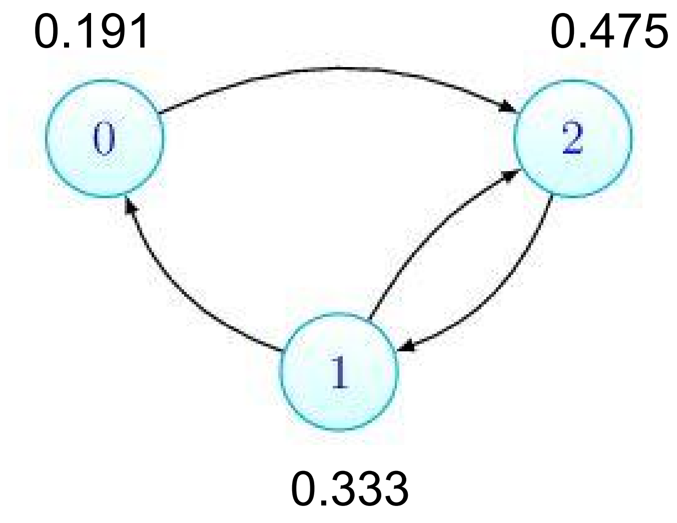
The PageRank Algorithm

Node 0 passes its score to Node 2.
Node 1 passes its score to Node 0 and Node 2.
Node 2 passes its score to Node 1.

Iteration 0



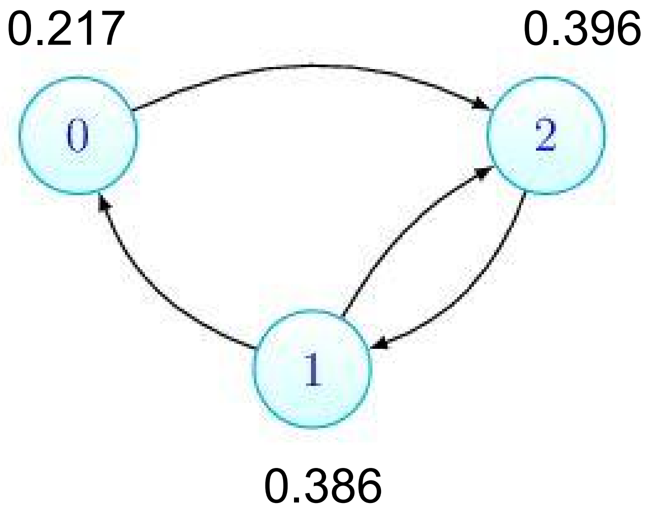
Iteration 1

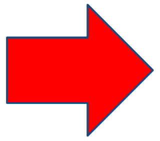


The PageRank Algorithm

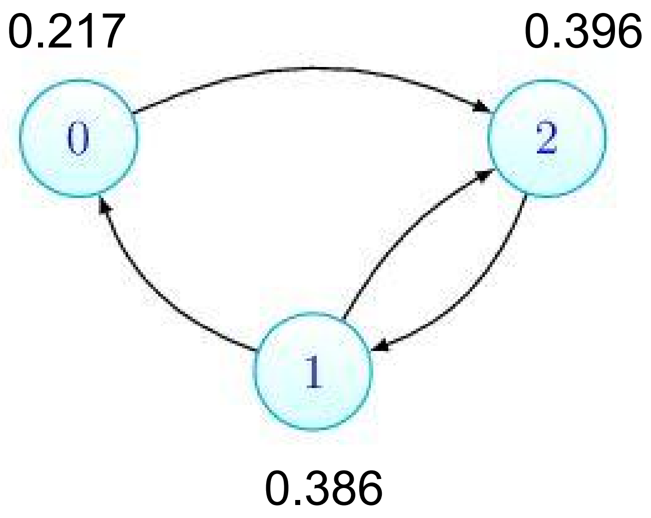
When the score of every node does not change across iterations, we refer to it as the algorithm converged.

Iteration k

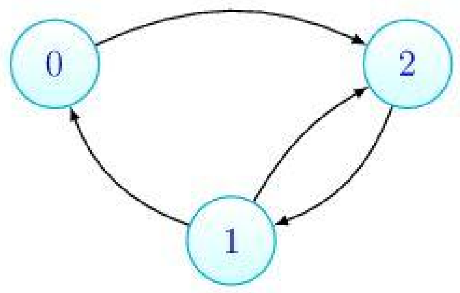




Iteration k+1



The PageRank Algorithm



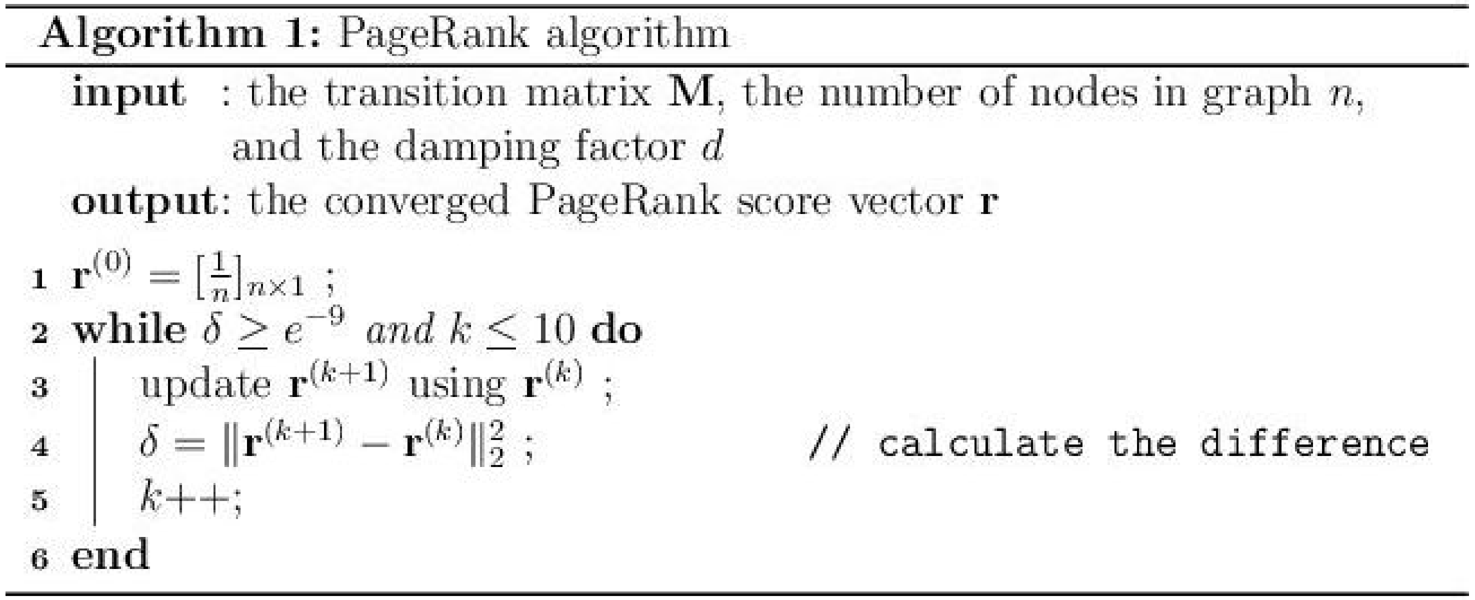
- Adjacent matrix: $\mathbf{G} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

- Transition matrix: (row sums to 1)

$$M_{ij} = \frac{G_{ij}}{\sum_{k=1}^{n} G_{ik}} (\text{ when } \sum_{k=1}^{n} G_{ik} \neq 0) \qquad \mathbf{M} = \begin{bmatrix} 0 & 0 & 1 \\ 0.5 & 0 & 0.5 \\ 0 & 1 & 0 \end{bmatrix}$$

What if a node has zero outgoing links?
Set each element of that row to 1/n,
where n is the number of nodes in the graph.

The PageRank Algorithm



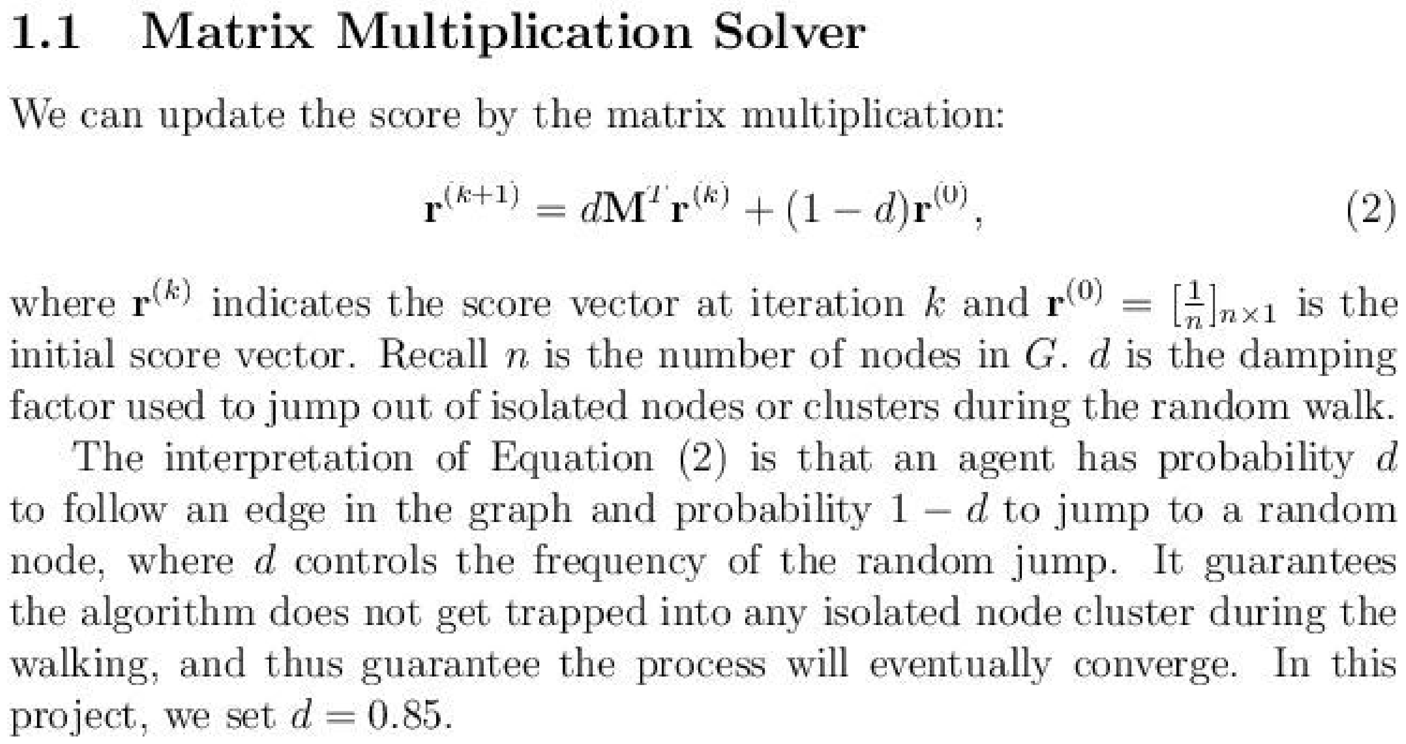
Algorithm 1: PageRank algorithm

```
input : the transition matrix $\mathbf{M}$, the number of nodes in graph $n$,
        and the damping factor $d$
output: the converged PageRank score vector $\mathbf{r}$
1 $\mathbf{r}^{(0)} = [\frac{1}{n}]_{n \times 1}$ ;
2 while $\delta \geq e^{-9}$ and $k \leq 10$ do
3     update $\mathbf{r}^{(k+1)}$ using $\mathbf{r}^{(k)}$ ;
4     $\delta = \|\mathbf{r}^{(k+1)} - \mathbf{r}^{(k)}\|_2^2$ ;          // calculate the difference
5     $k$++;
6 end
```

PageRank algorithm can usually converge in 10 iterations.
Two ways to implement Step 3:

- matrix solver
- for-loop solver (required)

Matrix Solver



1.1 Matrix Multiplication Solver

We can update the score by the matrix multiplication:

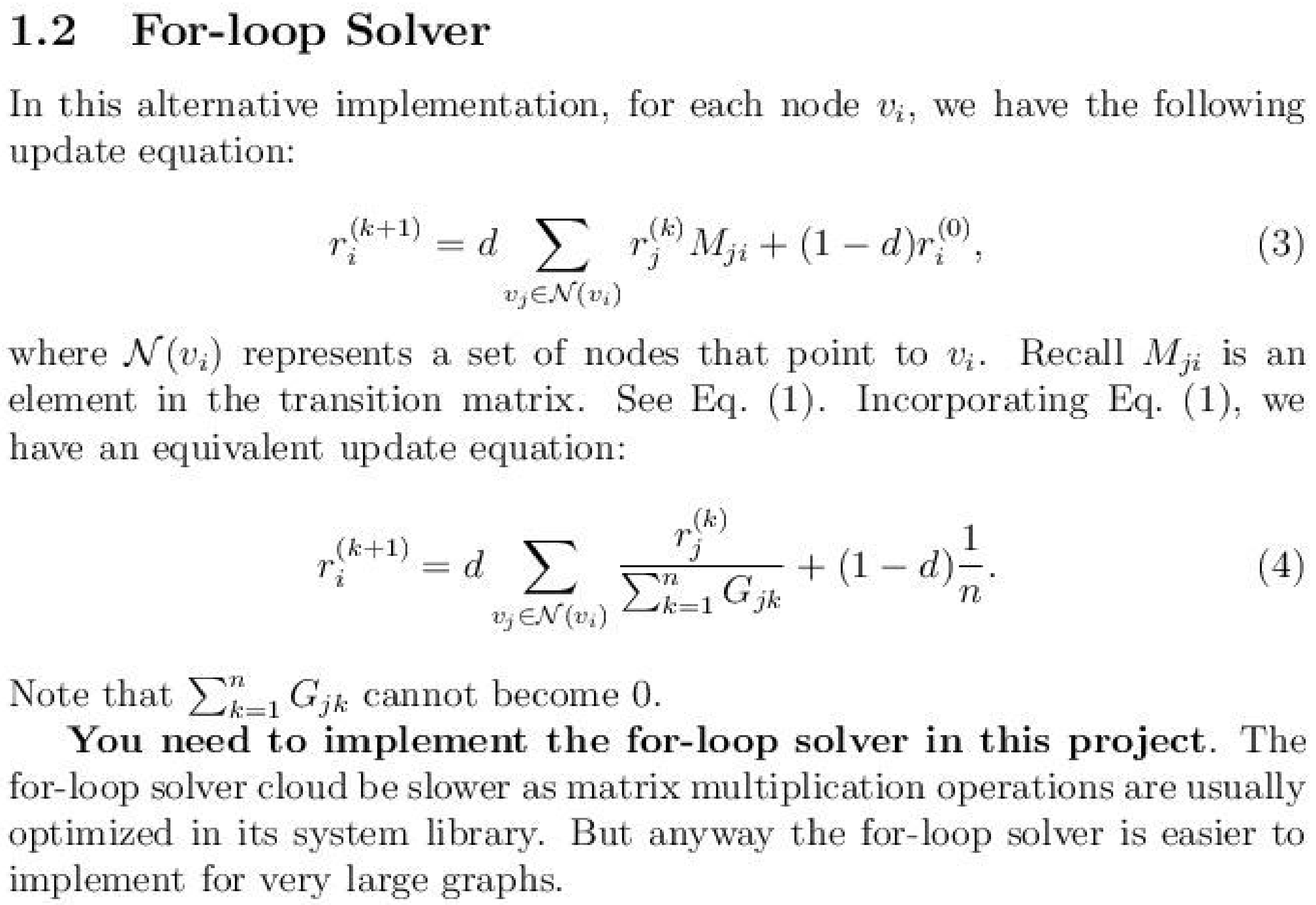
$$\mathbf{r}^{(k+1)} = d\mathbf{M}^T\mathbf{r}^{(k)} + (1-d)\mathbf{r}^{(0)}, \tag{2}$$

where $\mathbf{r}^{(k)}$ indicates the score vector at iteration k and $\mathbf{r}^{(0)} = [\frac{1}{n}]_{n\times 1}$ is the initial score vector. Recall n is the number of nodes in G. d is the damping factor used to jump out of isolated nodes or clusters during the random walk.

The interpretation of Equation (2) is that an agent has probability d to follow an edge in the graph and probability $1-d$ to jump to a random node, where d controls the frequency of the random jump. It guarantees the algorithm does not get trapped into any isolated node cluster during the walking, and thus guarantee the process will eventually converge. In this project, we set $d = 0.85$.

- Matrix implementation can be very fast. But it might not be applied to very large graphs due to the memory constraint.
- Use sparse matrix to store M.

For-loop Solver



1.2 For-loop Solver

In this alternative implementation, for each node v_i, we have the following update equation:

$$r_i^{(k+1)} = d \sum_{v_j \in \mathcal{N}(v_i)} r_j^{(k)} M_{ji} + (1-d) r_i^{(0)}, \tag{3}$$

where $\mathcal{N}(v_i)$ represents a set of nodes that point to v_i. Recall M_{ji} is an element in the transition matrix. See Eq. (1). Incorporating Eq. (1), we have an equivalent update equation:

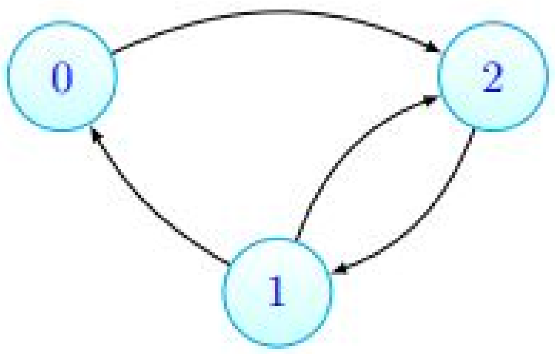
$$r_i^{(k+1)} = d \sum_{v_j \in \mathcal{N}(v_i)} \frac{r_j^{(k)}}{\sum_{k=1}^{n} G_{jk}} + (1-d)\frac{1}{n}. \tag{4}$$

Note that $\sum_{k=1}^{n} G_{jk}$ cannot become 0.

You need to implement the for-loop solver in this project. The for-loop solver cloud be slower as matrix multiplication operations are usually optimized in its system library. But anyway the for-loop solver is easier to implement for very large graphs.

- Less efficient but scalable to very large graphs.
- You are required to implement the for-loop solver.

A Toy Example - 1


$$\begin{tikzcd} 0 \arrow[rr, bend left] & & 2 \arrow[dl, bend left] \\ & 1 \arrow[ul, bend left] \arrow[ur, bend left] & \end{tikzcd}$$

Adjacent matrix $\mathbf{G}$:

$$\mathbf{G} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

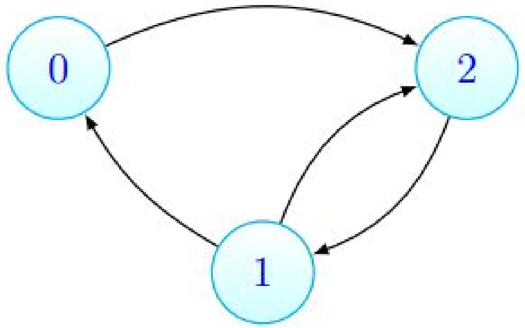
Transition matrix $\mathbf{M}$

$$\mathbf{M} = \begin{bmatrix} 0 & 0 & 1 \\ 0.5 & 0 & 0.5 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\boxed{M_{ij} = \frac{G_{ij}}{\sum_{k=1}^{n} G_{ik}} (\text{ when } \sum_{k=1}^{n} G_{ik} \neq 0)}$$

A Toy Example - 1

$$r_i^{(k+1)} = d \sum_{v_j \in \mathcal{N}(v_i)} r_j^{(k)} M_{ji} + (1-d) r_i^{(0)},$$



$d = 0.85$

$$\mathbf{M} = \begin{bmatrix} 0 & 0 & 1 \\ 0.5 & 0 & 0.5 \\ 0 & 1 & 0 \end{bmatrix}$$

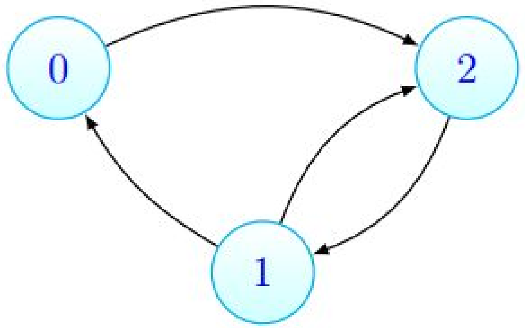
$$r_0^{(1)} = d\frac{r_1^{(0)}}{2} + (1-d)\frac{1}{n}$$

$$r_1^{(1)} = d\frac{r_2^{(0)}}{1} + (1-d)\frac{1}{n}$$

$$r_2^{(1)} = d(\frac{r_0^{(0)}}{1} + \frac{r_1}{2}) + (1-d)\frac{1}{n}$$

A Toy Example - 1

$$\boxed{r_i^{(k+1)} = d \sum_{v_j \in \mathcal{N}(v_i)} r_j^{(k)} M_{ji} + (1-d) r_i^{(0)},}$$



$d = 0.85$

$$\mathbf{M} = \begin{bmatrix} 0 & 0 & 1 \\ 0.5 & 0 & 0.5 \\ 0 & 1 & 0 \end{bmatrix}$$

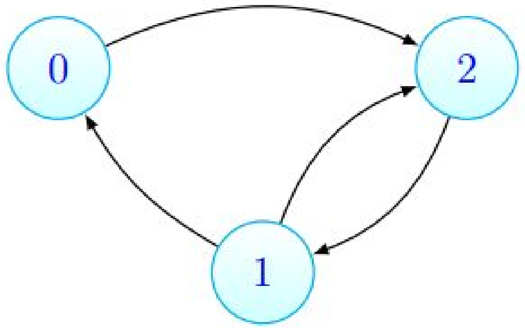
$$r_0^{(1)} = 0.85\frac{1}{6} + 0.15\frac{1}{3} = 0.191$$

$$r_1^{(1)} = 0.85\frac{1}{3} + 0.15\frac{1}{3} = 0.333$$

$$r_2^{(1)} = 0.85(\frac{1}{3} + \frac{1}{6}) + 0.15\frac{1}{3} = 0.475$$

A Toy Example - 1

$$\boxed{r_i^{(k+1)} = d \sum_{v_j \in \mathcal{N}(v_i)} r_j^{(k)} M_{ji} + (1-d) r_i^{(0)},}$$

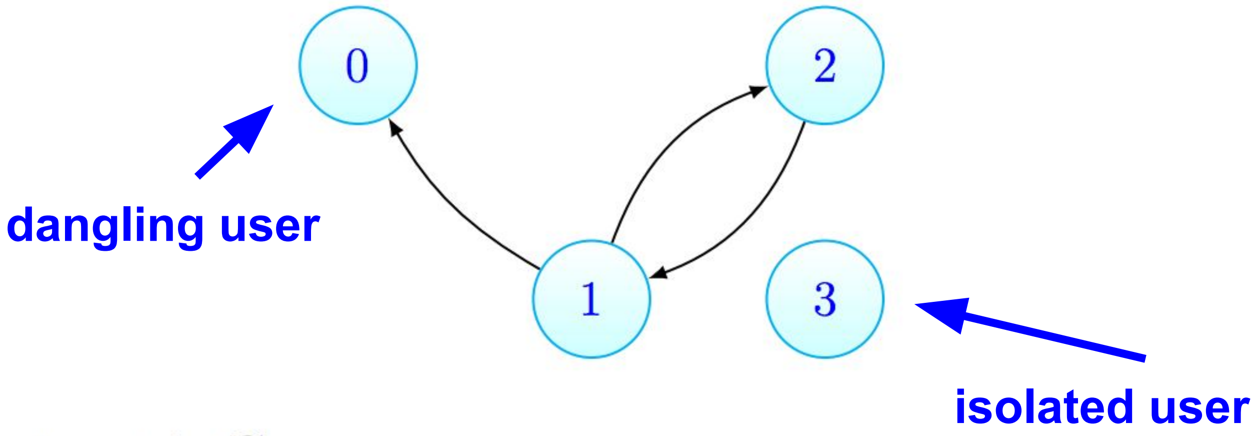


$$\mathbf{M} = \begin{bmatrix} 0 & 0 & 1 \\ 0.5 & 0 & 0.5 \\ 0 & 1 & 0 \end{bmatrix}$$

The converged result is

$$r_0^{(k)} = 0.217$$
$$r_1^{(k)} = 0.386$$
$$r_2^{(k)} = 0.396$$

A Toy Example - 2



Adjacent matrix $\mathbf{G}$:

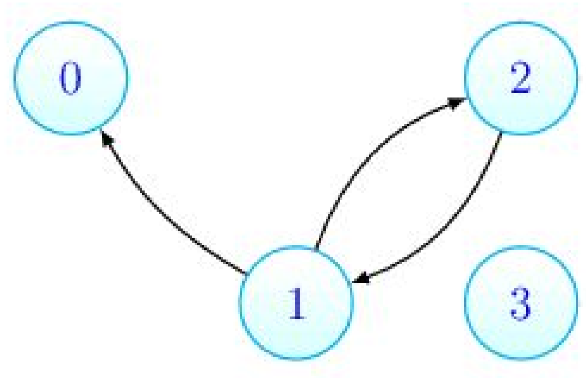
$$\mathbf{G} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Transition matrix $\mathbf{M}$

$$\mathbf{M} = \begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix}$$

A Toy Example - 2

$$r_i^{(k+1)} = d \sum_{v_j \in \mathcal{N}(v_i)} r_j^{(k)} M_{ji} + (1-d) r_i^{(0)},$$



$$r_0^{(1)} = d(\frac{r_1^{(0)}}{2} + \frac{r_0^{(0)}}{4} + \frac{r_3^{(0)}}{4}) + (1-d)\frac{1}{n}$$

$$r_1^{(1)} = d(\frac{r_2^{(0)}}{1} + \frac{r_0^{(0)}}{4} + \frac{r_3^{(0)}}{4}) + (1-d)\frac{1}{n}$$

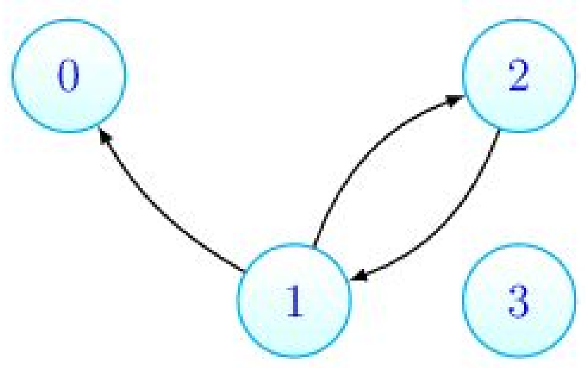
$$r_2^{(1)} = d(\frac{r_1^{(0)}}{2} + \frac{r_0^{(0)}}{4} + \frac{r_3^{(0)}}{4}) + (1-d)\frac{1}{n}$$

$$r_3^{(1)} = d(\frac{r_0^{(0)}}{4} + \frac{r_3^{(0)}}{4}) + (1-d)\frac{1}{n}$$

$$\mathbf{M} = \begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix}$$

A Toy Example - 2

$$r_i^{(k+1)} = d \sum_{v_j \in \mathcal{N}(v_i)} r_j^{(k)} M_{ji} + (1-d) r_i^{(0)},$$



$$\epsilon = d\left(\frac{r_0^{(0)}}{4} + \frac{r_3^{(0)}}{4}\right) \qquad d = 0.85$$

$$\begin{aligned}
\epsilon &= 0.85 \times (0.25/4 + 0.25/4) = 0.106 \\
r_0^{(1)} &= 0.85 \times 0.25/2 + 0.106 + 0.15 \times 0.25 = 0.25 \\
r_1^{(1)} &= 0.85 \times 0.25 + 0.106 + 0.15 \times 0.25 = 0.356 \\
r_2^{(1)} &= 0.85 \times 0.25/2 + 0.106 + 0.15 \times 0.25 = 0.25 \\
r_3^{(1)} &= 0.106 + 0.15 \times 0.25 = 0.144
\end{aligned}$$

PageRank in Spark (Scala)

(Note: This is a pseudocode of PageRank, simpler than P4.2)

```
val links = spark.textFile(...).map(...).persist()
var ranks = // RDD of (URL, rank) pairs
for (i <- 1 to ITERATIONS)
{
    // Build an RDD of (targetURL, float) pairs
    // with the contributions sent by each page
    val contribs = links.join(ranks).flatMap
    {
        (url, (links, rank)) =>
        links.map(dest => (dest, rank/links.size))
    }

    // Sum contributions by URL and get new ranks
    ranks = contribs.reduceByKey((x,y) => x+y)
                    .mapValues(sum => a/N + (1-a)*sum)
}
```

Bonus Task: Speed up Spark

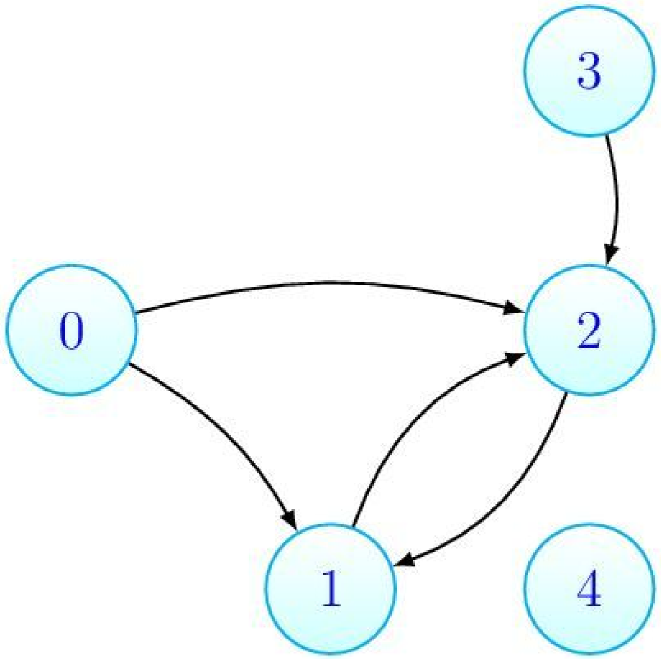
Hints:

- Eliminate repeated calculations in PageRank.
- Monitor your instances to make sure they are fully utilized.
- Develop a better understanding of RDD manipulations. Understand the "lazy" transformation in Spark.
- Spark is a tunable framework where there are many parameters that you can configure to make the best use of the resources.
- Be careful with repartition on your RDDs.

Graph Processing using GraphX

Task 3: Second-degree Centrality

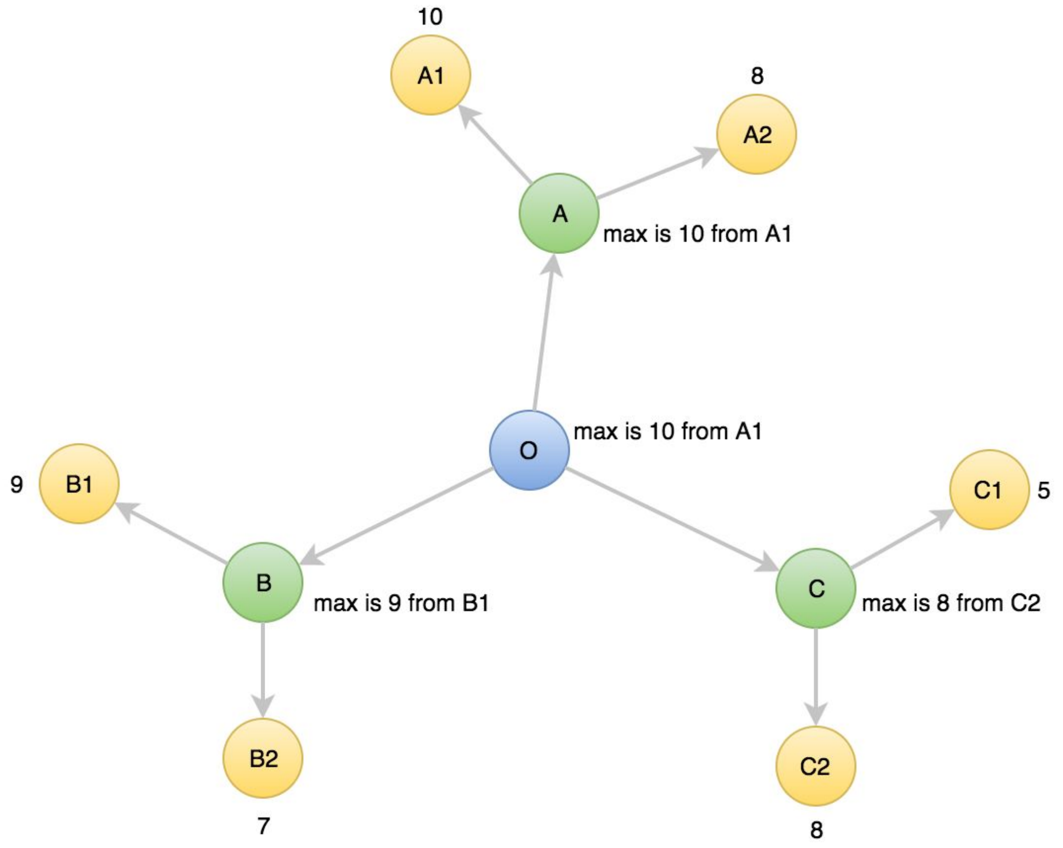
- PageRank score is a type of centrality score.
  - Importance of a node in a graph.
- However, the PageRank score for Node0 and Node4 is the same: 0.0306.
  - Does not make sense!
- Use PageRank to measure a 2nd degree centrality score.



Graph Processing using GraphX

Task 3: Second-degree Centrality

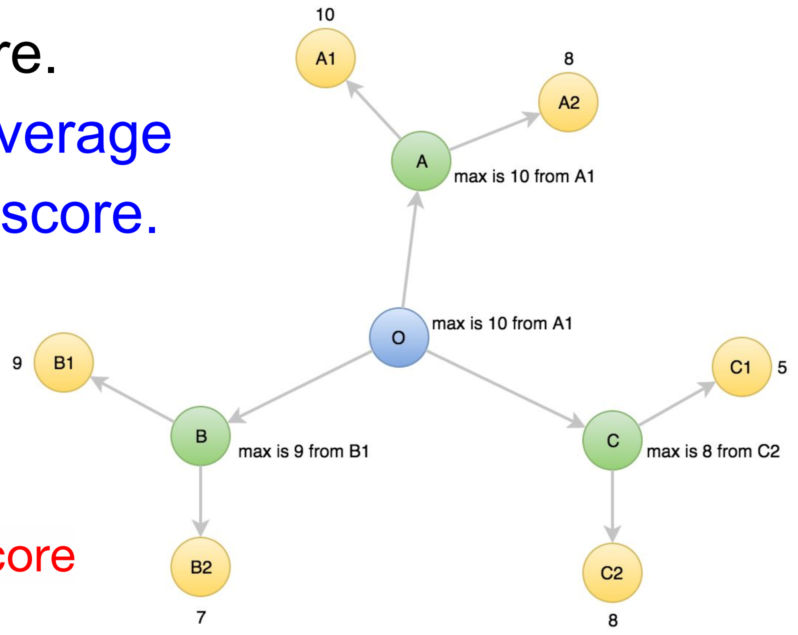
- From all the people your followees follow (i.e. your 2nd degree followees), find the 2nd-degree centrality score which is the highest PageRank score within the reach of 1 or 2 jumps.



Graph Processing using GraphX

Task 3: Second-degree Centrality

- From all the people your followees follow (i.e. your 2nd degree followees), find the one with the highest PageRank score.
- First calculate this score and then average this score with its original pagerank score.



new_influencial_score = 0.5 * pagerank_score + 0.5 * most_influential_second_degree_user_score

Hints for Launching a Spark Cluster

- Spark is an in-memory system
  - Develop and test your scripts on a portion of the dataset before launching a big cluster.
  - Look at examples in the writeups.
- A regular run in Task 2 would take 45 min using **5** r3.xlarge.
- Spark can run on your laptop. It is possible to test it on your laptop.

Spark Shell

- Like the python shell
- Run commands interactively
- On the master, execute (from /root)
  - ./spark/bin/spark-shell
  - ./spark/bin/pyspark

P4.2 Grading - 1

- Submit your work in the submitter instance
- Don’t forget to submit your code
- For Task 1
    - Put the number of nodes and edges in the answer file
    - Put your Spark program that count the number of followers for each user in the given folder
    - Run the submitter to submit
- For Task 2
    - Put your Spark program that calculates the pagerank score for each user in the given folder
    - Run the submitter to submit
    - Check if the sum of PageRank score is 1.0 at each iteration
    - **Bonus**: tune your system to run as fast as possible

P4.2 Grading - 2

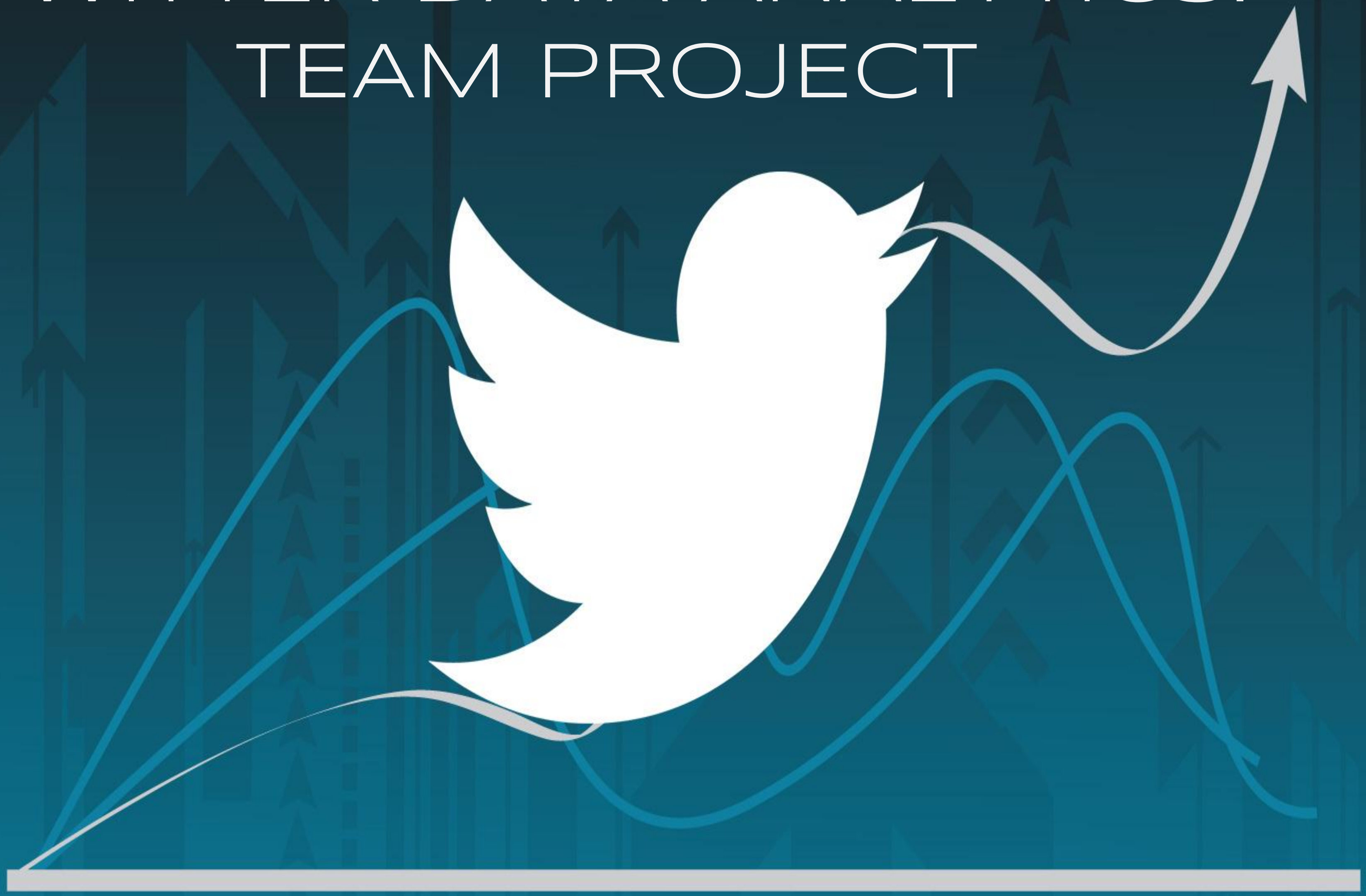
- Submit your work in the submitter instance
- Don’t forget to submit your code
- For Task 3
  - Put your GraphX program to calculate the new centrality score in the given folder
  - Run the submitter to submit

Upcoming Deadlines

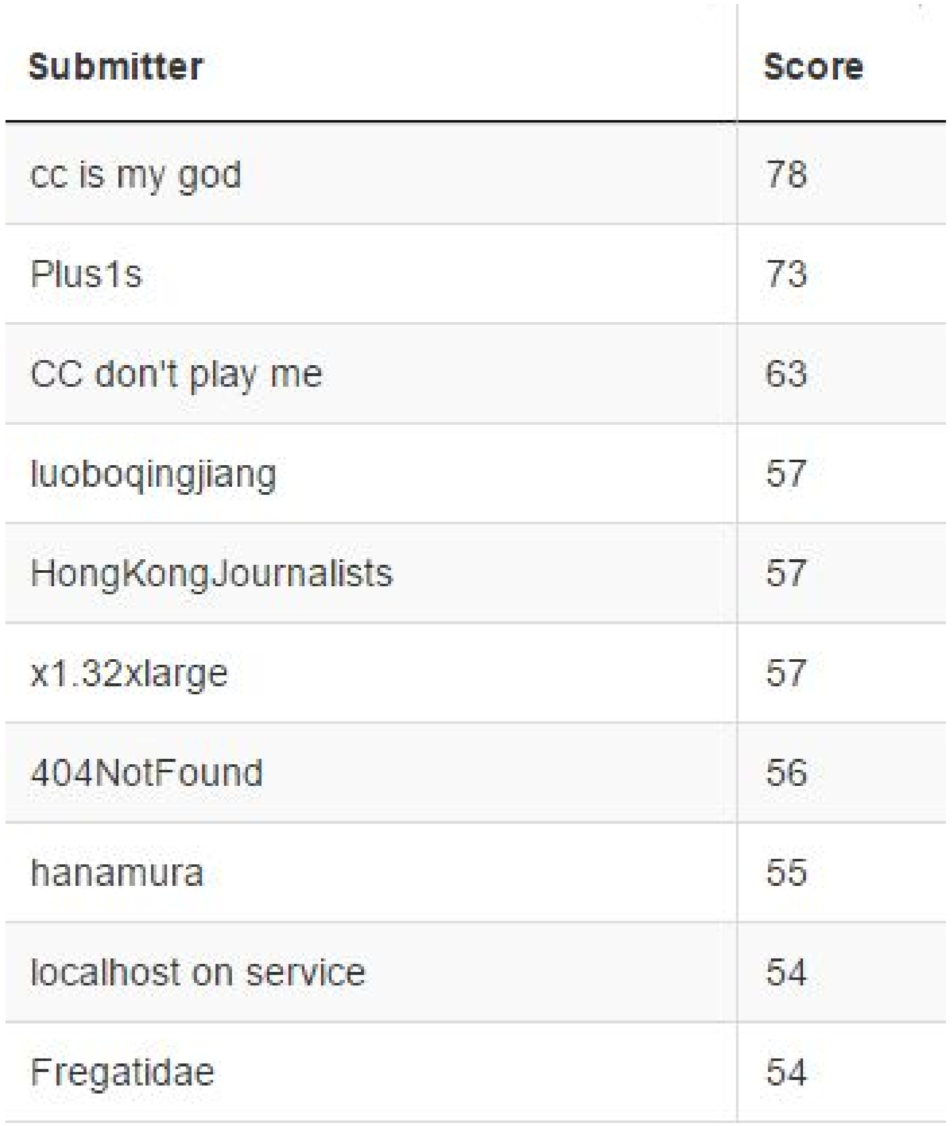
- Team Project : Phase 2
  - Code and report due: 04/18/2017 11:59 PM Pittsburgh
- Project 4.2 : Iterative Programming with Spark
  - Due: 04/23/2017 11:59 PM Pittsburgh
- Team Project : Phase 3
  - Live-test due: 04/30/2017 3:59 PM Pittsburgh
  - Code and report due: 05/02/2017 11:59 PM Pittsburgh

Questions?

TWITTER DATA ANALYTICS: TEAM PROJECT

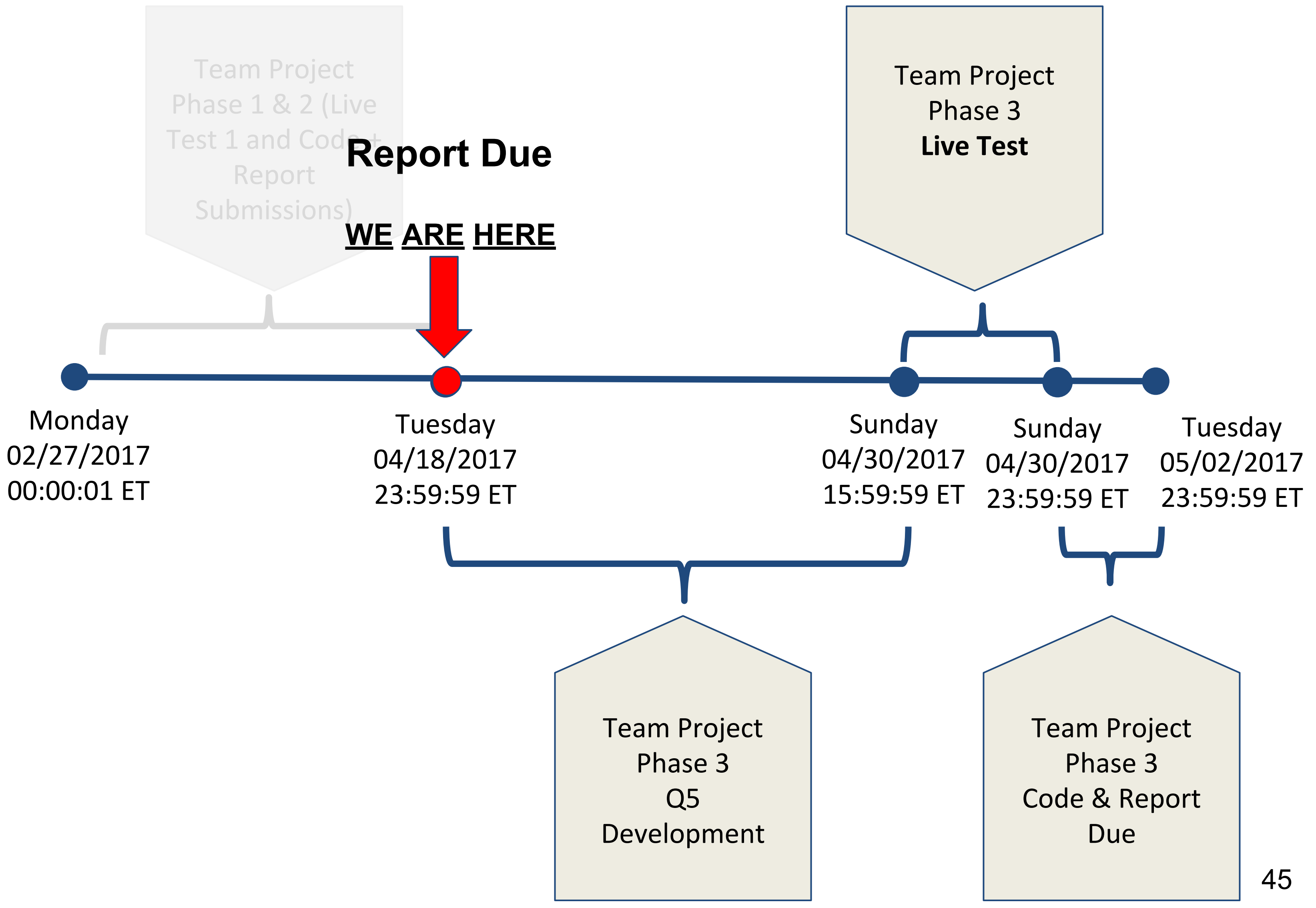


Team Project Phase 2 Live Test Honor Board



Submitter	Score
cc is my god	78
Plus1s	73
CC don't play me	63
luoboqingjiang	57
HongKongJournalists	57
x1.32xlarge	57
404NotFound	56
hanamura	55
localhost on service	54
Fregatidae	54

Team Project Phase 3 Deadlines


Team Project Phase 1 & 2 (Live Test 1 and Code + Report Submissions)
Report Due
WE ARE HERE
Team Project Phase 3 Live Test
Monday 02/27/2017 00:00:01 ET
Tuesday 04/18/2017 23:59:59 ET
Sunday 04/30/2017 15:59:59 ET
Sunday 04/30/2017 23:59:59 ET
Tuesday 05/02/2017 23:59:59 ET
Team Project Phase 3 Q5 Development
Team Project Phase 3 Code & Report Due

Team Project, Query 5

- Building a graph using the “mention” in the tweets.
  - Users POST Tweets, Tweets MENTION Users.

- Finding the shortest path between two users in this “mention” graph.

Team Project, Query 5

- This is a **directed** graph!
- There could be multiple shortest paths from starting node to ending node. We break ties using the lowest TID and UID.
  - Assuming there are two shortest paths
    - UID:1-->**TID:10**-->UID:2 (Correct output)
    - UID:1-->**TID:20**-->UID:2 (Wrong output)
  - And we break ties using the lowest UID, if a tweet has mentioned multiple users.
    - UID:1-->TID:10-->**UID:3**-->TID:30-->UID:2 (Correct output)
    - UID:1-->TID:10-->**UID:4**-->TID:20-->UID:2 (Wrong output)

Team Project Time Table

Phase (and query due)	**Start**	**Deadline**	**Code and Report Due**
Phase 1 ● Q1, Q2	Monday 10/10/2016 00:00:01 EST	Sunday 10/30/2016 23:59:59 ET	**Tuesday 11/01/2016 23:59:59 ET**
Phase 2 ● Q1, Q2, Q3, Q4	Monday 10/31/2016 00:00:01 ET	Sunday 11/13/2016 **15:59:59 ET**	
Phase 2 Live Test (Hbase/MySQL) ● Q1, Q2, Q3, Q4	Sunday 11/13/2016 18:00:01 ET	Sunday 11/13/2016 23:59:59 ET	**Tuesday 04/18/2017 23:59:59 ET**
Phase 3 ● Q1, Q2, Q3, Q4, Q5	**Monday 04/17/2017 00:00:01 ET**	**Sunday 04/30/2017 15:59:59 ET**	
Phase 3 Live Test ● Q1, Q2, Q3, Q4, Q5	Sunday 04/30/2017 18:00:01 ET	Sunday 04/30/2017 23:59:59 ET	Tuesday 05/02/2017 23:59:59 ET

Questions?