COMP9313: Big Data Management

Revisit and Sample Exam

Sample Exam paper is included in the revisited part.
Note for Question 1, 2, …

MyExperience Survey

- The UNSW myExperience survey is still open. Please submit your feedback.
- *"Please participate in the myExperience Survey and take the opportunity to share your constructive thoughts on your learning experience. Your contributions help your teachers and shape the future of education at UNSW."*
- You can access the survey by logging into Moodle or accessing myexperience.unsw.edu.au directly.
- **If the response rate from the class is more than 50%, everybody gets 1 bonus mark added to the final mark :-)**

Final exam

- **Time:** Fri 16-Aug, 8 am - 12 pm, 4 hours (Do not wait for the last minute to submit)
- **Exam paper:** will be released on our course website (Moodle and Ed) around 7:55 am on the exam day, allowing you for an extra 5 min to download the paper and upload your solutions.
- **How to submit:**
- Coding question submitted through Ed like project, and others through Moodle in one .pdf or .doc file.
- You can submit multiple times, and we will mark the last one.

Final exam

- Final written exam (50 pts)
- **Double Pass:** You also need to achieve at least 20 marks in the final exam to pass the course.
- 6 questions in total on 5 topics
- There will be consultations before the final exam. Detailed schedule will be released later.
- **Special Considerations: The exam is covered by UNSW's Fit-to-Sit policy. That means that by sitting this exam, you are declaring yourself well enough to do so. You will be unable to apply for special consideration after the exam for circumstances affecting you before it began.**

Overview

- Hadoop MapReduce
  - HDFS
  - MapReduce Concepts and Mechanism
  - MapReduce algorithm design
- Spark
  - RDD
  - DataFrame
- Mining Data Streams
- Finding Similar Items
  - Shingling, Minhash, LSH
- Graph Data Management

Exam Questions

- Question 1: HDFS, MapReduce, and Spark concepts
- Question 2: MapReduce algorithm design (pseudo-code only)
- Question 3: Spark algorithm design
  - RDD
  - DataFrame
- Question 4 Finding Similar Items
- Question 5 Mining Data Streams
- Question 6 Graph Data Management

Sample Question 1

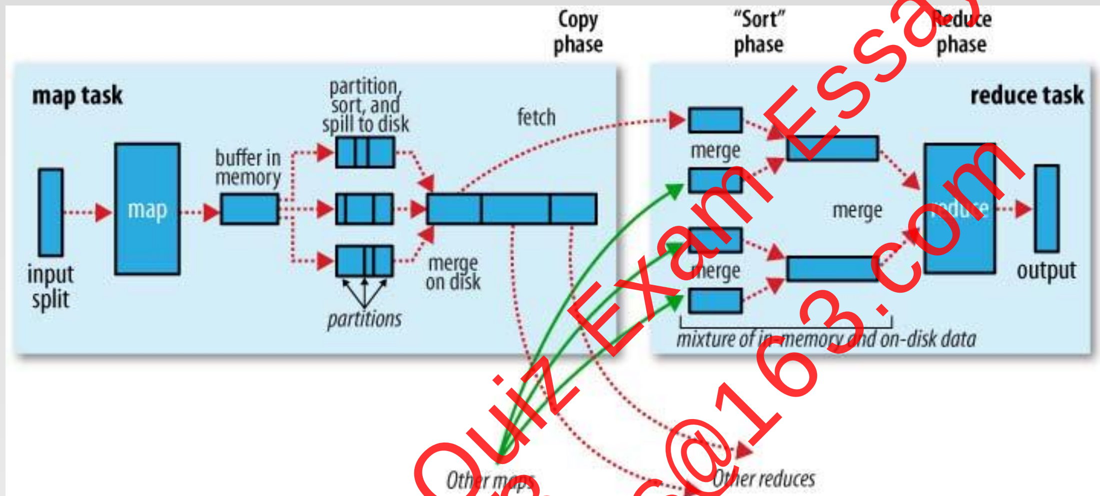
- (a) (2 marks) Explain the data flow in MapReduce using the word count problem as an example.

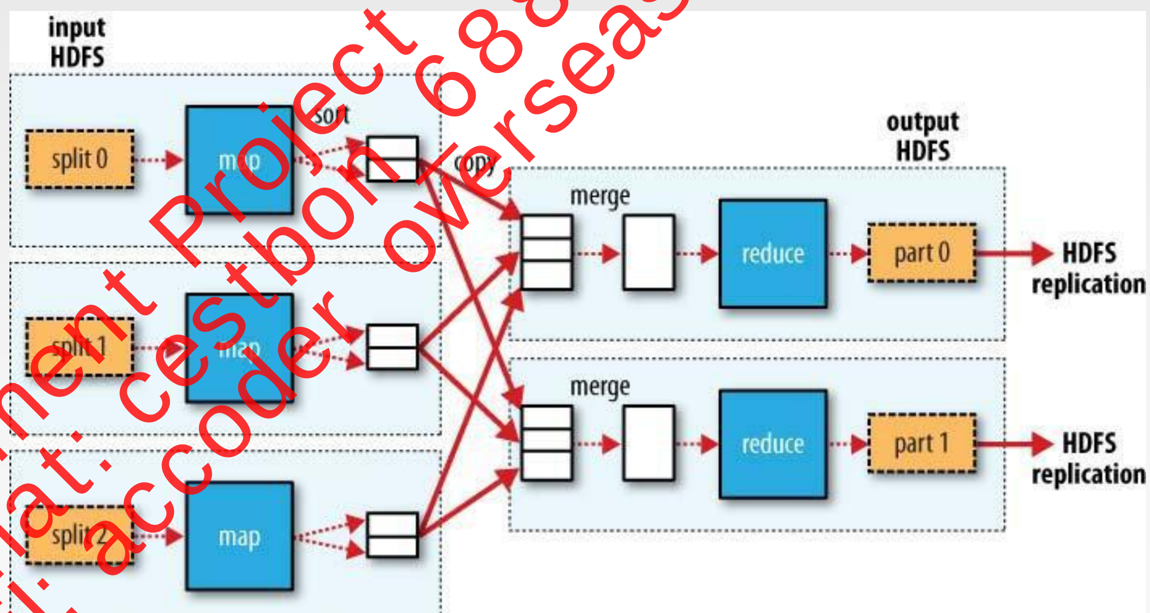
- (b) (2 marks) Explain the data flow in Spark using the word count problem as an example.

Map and Reduce Functions

- Programmers specify two functions:
  - **map** $(k_1, v_1) \rightarrow$ list $[<k_2, v_2>]$
    - Map transforms the input into key-value pairs to process
  - **reduce** $(k_2, [v_2]) \rightarrow [<k_3, v_3>]$
    - Reduce aggregates the list of values for each key
    - All values with the same key are sent to the same reducer
- Optionally, also:
  - combine $(k_2, [v_2]) \rightarrow [<k_3, v_3>]$
  - partition (k_2, number of partitions) $\rightarrow$ partition for k_2
  - Grouping comparator: controls which keys are grouped together for a single call to Reducer.reduce() function
- The execution framework handles everything else…

MapReduce Data Flow





Sample Question 2

- Assume that you are given a data set crawled from a location-based social network, in which each line of the data is in format of (userID, a list of locations the user has visited <loc1, loc2, …>). Your task is to compute for each location the set of users who have visited it, and the users are sorted in ascending order according to their IDs.

Sample Solution

```
class Question1
        method map(self, userID, list of locations)
                foreach loc in the list of locations
                          Emit(“loc, userID”, “”)

        method reduce_init(self)
                current_loc = “”
                current_list = []

        method reduce(self, key, value)
                loc, userID = key.split(“,”)
                if loc != current_loc
                      if current_loc!=“”
                           Emit(current_loc, current_list)
                      current_list = []
                      current_list.add(userID)
                      current_loc=loc
                else
                      current_list.add(userID)

         method reduce_final(self)
                Emit(current_loc, current_list)

In JOBCONF, configure:
         'mapreduce.map.output.key.field.separator':’,’,
         'mapreduce.partition.keypartitioner.options':'-k1,1’,
         'mapreduce.partition.keycomparator.options':'-k1,1 -k2,2'
```

Sample Question 2

- Given a table shown as below, find out the person(s) with the maximum salary in each department (employees could have the same salary).

EmployeeID	Name	DepartmentID	Salary
001	Emma	1	100,000
002	Helen	2	85,000
003	Jack	3	85,000
004	James	1	110,000

- Solution:
  - Mapper: for each record, Emit(department + "," + salary, name)
  - Combiner: find out all persons with the local maximum salary for each department
  - Reducer: receives data ordered by (department, salary), the first one is the maximum salary in a department. Check the next one until reaching a smaller salary and ignore all remaining. Save all persons with this maximum salary in the department
  - JOBCONF: key partitioned by "-k1,1", sorted by "-k1,1 -k2,2n"

- In the final, check the question requirement, asking for pseudo code or others…

What is RDD

- Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. Matei Zaharia, et al. NSDI’12
  - RDD is a **distributed** memory abstraction that lets programmers perform **in-memory** computations on large clusters in a **fault-tolerant** manner.
- **Resilient**
  - Fault-tolerant, is able to recompute missing or damaged partitions due to node failures.
- **Distributed**
  - Data residing on multiple nodes in a cluster.
- **Dataset**
  - A collection of partitioned elements, e.g. tuples or other objects (that represent records of the data you work with).
- RDD is the primary data abstraction in Apache Spark and the core of Spark. It enables operations on collection of elements in parallel.

DataFrame

- DataFrame more like a traditional database of two-dimensional form, in addition to data, but also to grasp the structural information of the data, that is, schema

  - RDD[Person] although with Person for type parameters, but the Spark framework itself does not understand internal structure of Person class
  - DataFrame has provided a detailed structural information, making Spark SQL can clearly know what columns are included in the dataset, and what is the name and type of each column. Thus, Spark SQL query optimizer can target optimization

Sample Question 3

- **RDD:** Given a large text file, your task is to find out the top-k most frequent co-occurring term pairs. The co-occurrence of (w, u) is defined as: u and w appear in the same line (this also means that (w, u) and (u, w) are treated equally). Your Spark program should generate a list of ***k*** key-value pairs ranked in descending order according to the frequencies, where the keys are the pair of terms and the values are the co-occurring frequencies (**Hint:** you need to define a function which takes an array of terms as input and generate all possible pairs).

```
val textFile = sc.textFile(inputFile)
val words = textFile.map(_.split(" ").toLowerCase)

// fill your code here, and store the result in a pair RDD topk

topk.foreach(x => println(x._1, x._2))
```

- *Note: python code is ok.*

Sample Question 3

- Given a set of marks from different courses (the input format is as shown in the left column), the task is to: compute average marks for every course and sort the result by course_name in alphabetical order.

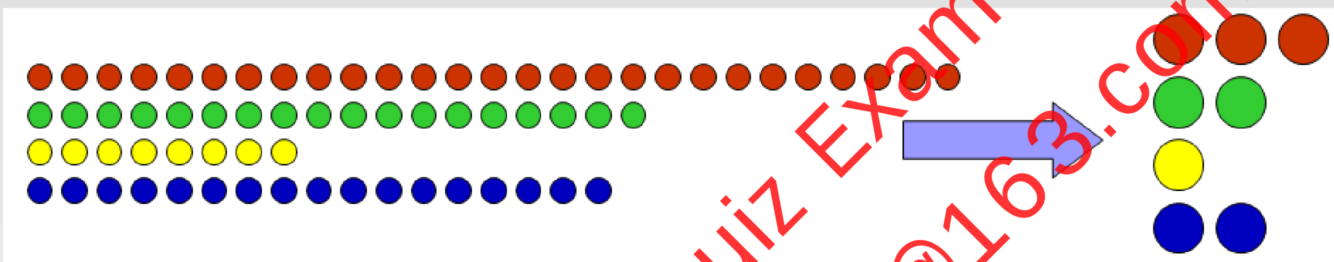
Input:	Output:
student1:course1,90;course2,92;course3,80;course4, 79;course5,93 student2:course1,92;course2,77;course5,85 student3:course3,64;course4,97;course5,82	course1:91 course2:84.5 course3:72 course4:88 course5:86.67

- Solution:

```
fileDF = spark.read.text("file:///home/comp9313/tinydoc")

student = fileDF.select(split(fileDF['value'], ':').getItem(0).alias('sid'), split(fileDF['value'],
':').getItem(1).alias('courses'))

scDF = student.withColumn('course', explode(split('courses', ';')))

scDF2 = scDF.select(split(scDF['course'], ',').getItem(0).alias('cname'), split(scDF['course'],
',').getItem(1).alias('mark'))

avgDF = scDF2.groupBy('cname').agg(avg('mark')).orderBy('cname')
```

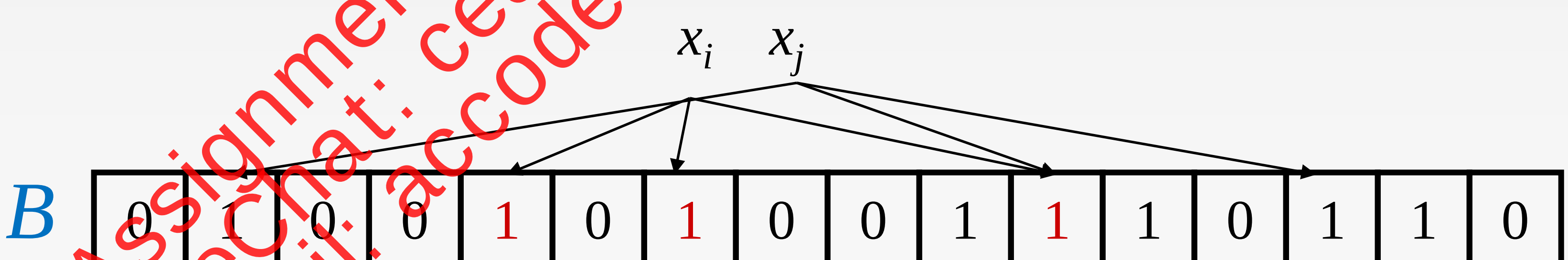
Mining Data Streams

- Sampling from a data stream



- Sliding window – counting bits (DGIM)

- Filtering data stream – bloom filter



Mining Data Streams

- Finding Frequent Elements
  - Boyer-Moore voting algorithm, Misra-Gries algorithm
  - count-min sketch
- Counting data stream – FM-Sketch
  - Estimate $d = c2^R$ for scaling constant $c \approx 1.3$ (original paper)

Sample Question 4

- Use an example to explain the reservoir sampling algorithm

- Store all the first **s** elements of the stream to **S**
- Suppose we have seen **n-1** elements, and now the $\mathbf{n}^{\mathbf{th}}$ element arrives (**n > s**)
  - With probability **s/n**, keep the $\mathbf{n}^{\mathbf{th}}$ element, else discard it
  - If we picked the $\mathbf{n}^{\mathbf{th}}$ element, then it replaces one of the **s** elements in the sample **S**, picked uniformly at random

Sample Question 4

Suppose we are maintaining a count of 1s using the DGIM method. We represent a bucket by (i, t), where i is the number of 1s in the bucket and t is the bucket timestamp (time of the most recent 1).

Consider that the current time is 200, window size is 60, and the current list of buckets is: (16, 148) (8, 162) (8, 177) (4, 183) (2, 192) (1, 197) (1, 200). At the next ten clocks, 201 through 210, the stream has 0101010101. What will the sequence of buckets be at the end of these ten inputs?

Sample Solution

- There are 5 1s in the stream. Each one will update to windows to be:
  - (1) (16, 148)(8, 162)(8, 177)(4, 183)(2, 192)(1, 197)(1, 200), (1, 202)

    => (16, 148)(8, 162)(8, 177)(4, 183)(2, 192)(2, 200), (1, 202)
  - (2) (16, 148)(8, 162)(8, 177)(4, 183)(2, 192)(2, 200), (1, 202), (1, 204)
  - (3) (16, 148)(8, 162)(8, 177)(4, 183)(2, 192)(2, 200), (1, 202), (1, 204), (1; 206)

    => (16, 148)(8, 162)(8, 177)(4, 183)(2, 192)(2, 200), (2, 204), (1, 206)

    => (16, 148)(8, 162)(8, 177)(4, 183)(4, 200), (2, 204), (1, 206)
  - (4) Windows Size is 60, so (16,148) should be dropped.

    (16, 148)(8, 162)(8, 177)(4, 183)(4, 200), (2, 204), (1, 206), (1, 208) => (8, 162)(8, 177)(4, 183)(4, 200), (2, 204), (1, 206), (1, 208)
  - (5) (8, 162)(8, 177)(4, 183)(4, 200), (2, 204), (1, 206), (1, 208), (1, 210)

    => (8, 162)(8, 177)(4, 183)(4, 200), (2, 204), (2, 208), (1, 210)

Sample Question 4

- Consider a Bloom filter of size m = 7 (i.e., 7 bits) and 2 hash functions that both take a string (lowercase) as input:

  $h1(str) = \sum_{(c \text{ in } str)}(c-'a') \bmod 7$

  $h2(str) = str.length \bmod 7$

Here, c - ‘a’ is used to compute the position of the letter c in the 26 alphabetical letters, e.g., h1(“bd”) = (1 + 3) mod 7 = 4.

- (i) Given a set of string S = {“hi”, “big”, “data”}, show the update of the Bloom filter
- (ii) Given a string “spark”, use the Bloom filter to check whether it is contained in S.
- (iii) Given S in (i) and the Bloom filter with 7 bits, what is the percentage of the false positive probability (a correct expression is sufficient: you need not give the actual number)?

Sample Solution

- (i)

	hi	big	data
h1	(7+8) mod 7 = 1	(1+8+6) mod 7 = 1	(3+0+19+0) mod 7 = 1
h2	2 mod 7 = 2	3 mod 7 = 3	4 mod 7 = 4

- (ii) h1 (spark) = (18 + 15 + 0 + 17 + 10) mod 7 = 4

  h2 (spark) = 5 mod 7 = 5

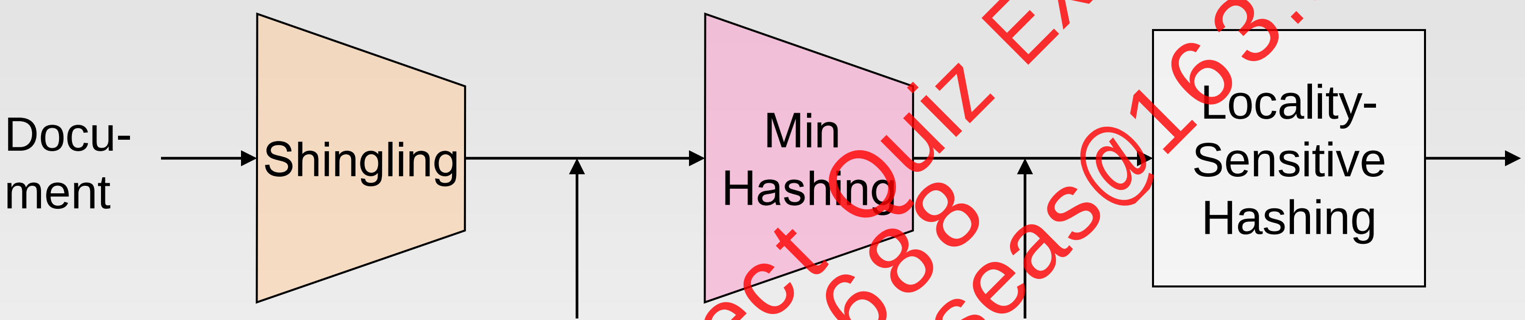
  Not in S since the 4th bit is 1 but the 5th bit is 0

- (iii) k – # of hash functions; m – # of inserting elements; n - # of bits

$$\left(\mathbf{1} - e^{-\frac{km}{n}}\right)^{k} = \mathbf{0.3313}$$

Finding Similar Items

❖ The Big Picture



The set of strings of length ***k*** that appear in the doc-ument

Signatures: short integer vectors that represent the sets, and reflect their similarity

Candidate pairs: those pairs of signatures that we need to test for similarity

Sample Question 5

We want to compute min-hash signature for two columns, C_1 and C_2 using two pseudo-random permutations of columns using the following function:

$$h_1(n) = 3n + 2 \bmod 7$$

$$h_2(n) = 2n - 1 \bmod 7$$

Row	C_1	C_2
0	0	1
1	1	0
2	0	1
3	0	0
4	1	1
5	1	1
6	1	0

Here, n is the row number in original ordering. Instead of explicitly reordering the columns for each hash function, we use the implementation discussed in class, in which we read each data in a column once in a sequential order, and update the min hash signatures as we pass through them.

Complete the steps of the algorithm and give the resulting signatures for C_1 and C_2.

Solution

Row	C_1	C_2
0	0	1
1	1	0
2	0	1
3	0	0
4	1	1
5	1	1
6	1	0

$h_1(n) = 3n + 2 \bmod 7$
$h_2(n) = 2n - 1 \bmod 7$

	Sig1	Sig2
	∞	∞
	∞	∞
h1(0) = 2	∞	2
h2(0) = 6	∞	6
h1(1) = 5	5	2
h2(1) = 1	1	6
h1(2) = 1	5	1
h2(2) = 3	1	3
h1(4) = 0	0	0
h2(4) = 0	0	0

Sample Question 5

- Suppose we wish to find similar sets, and we do so by minhashing the sets 10 times and then applying locality-sensitive hashing using 5 bands of 2 rows (minhash values) each. If two sets had Jaccard similarity 0.6, what is the probability that they will be identified in the locality-sensitive hashing as candidates (i.e. they hash at least once to the same bucket)? You may assume that there are no coincidences, where two unequal values hash to the same bucket. A correct expression is sufficient: you need not give the actual number.

- Solution: **$1 - (1 - t^r)^b$**
  - $1 - (1 - 0.6^2)^5$

Graph – Shortet path (iteration 1)

- Map:

Read s --> 0 | n1: 10, n2: 5

Emit: (n1, 10), (n2, 5), and the adjacency list (s, n1: 10, n2: 5)

The other lists will also be read and emit, but they do not contribute, and thus ignored

- Reduce:

Receives: (n1, 10), (n2, 5), (s, <0, (n1: 10, n2: 5)>)

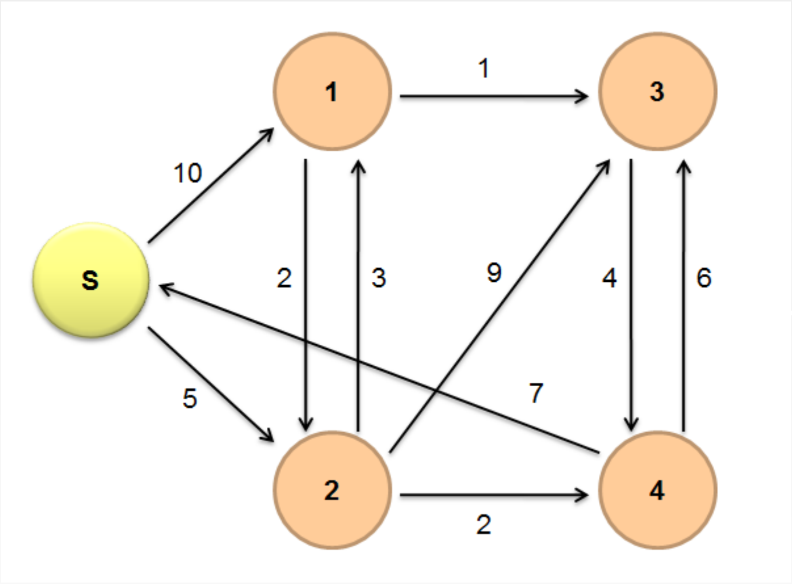
The adjacency list of each node will also be received, ignored in example

Emit:

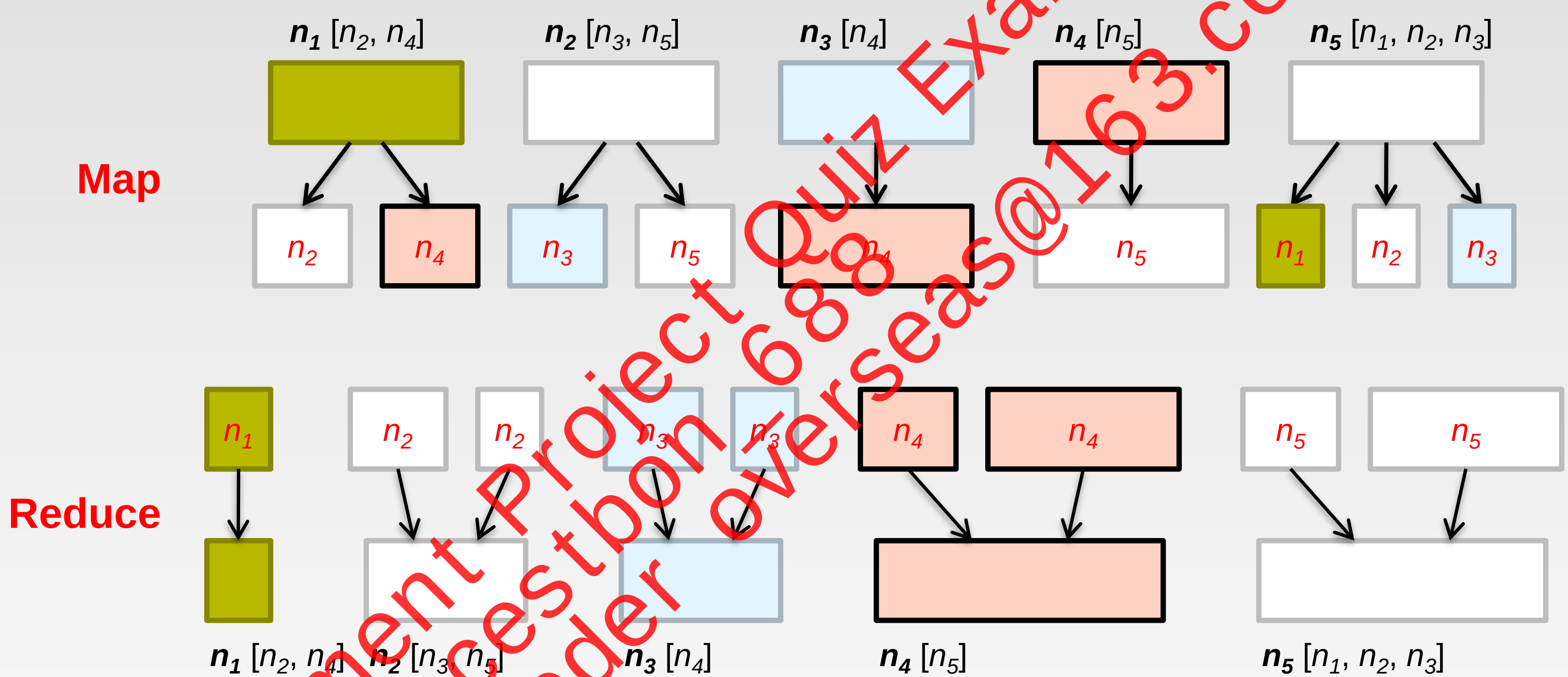
s --> 0 | n1: 10, n2: 5

n1 --> 10 | n2: 2, n3:1

n2 --> 5 | n1: 3, n3:9, n4:2

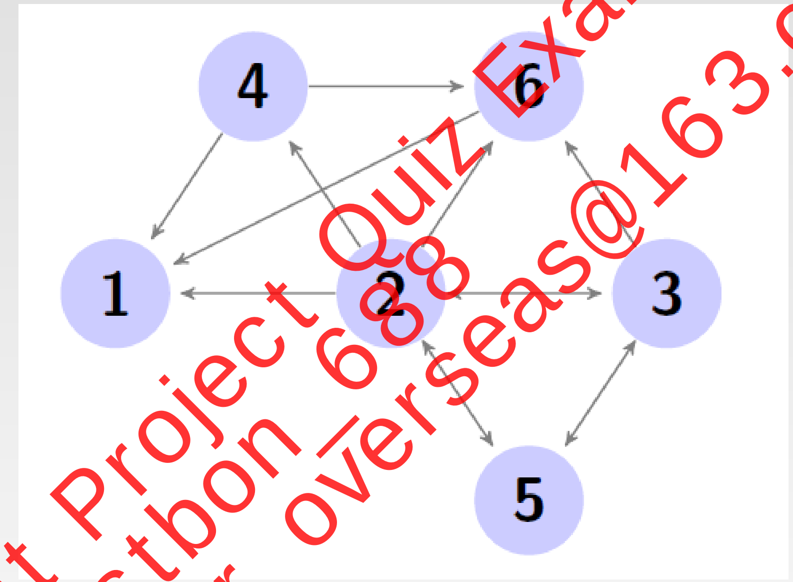


PageRank in MapReduce (One Iteration)



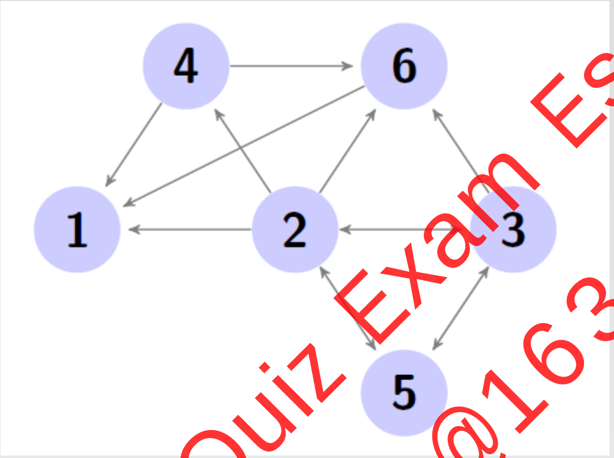
Sample Question 6

- A directed graph G has the set of nodes {1,2,3,4,5,6} with the edges arranged as follows.



- Set up the PageRank equations, assuming $\beta = 0.8$ (jump probability = $1- \beta$). Denote the PageRank of node a by r(a).

Solution



$$r(1) = 0.8(\frac{1}{6} \cdot r(1) + \frac{1}{2} \cdot r(4) + r(6) + \frac{1}{5} \cdot r(2)) + \frac{0.2}{6} \quad (1)$$

$$r(2) = 0.8(\frac{1}{6} \cdot r(1) + \frac{1}{3} \cdot r(3) + \frac{1}{2} \cdot r(5)) + \frac{0.2}{6} \quad (2)$$

$$r(3) = 0.8(\frac{1}{6} \cdot r(1) + \frac{1}{5} \cdot r(2) + \frac{1}{2} \cdot r(5)) + \frac{0.2}{6} \quad (3)$$

$$r(4) = 0.8(\frac{1}{6} \cdot r(1) + \frac{1}{5} \cdot r(2)) + \frac{0.2}{6} \quad (4)$$

$$r(5) = 0.8(\frac{1}{6} \cdot r(1) + \frac{1}{5} \cdot r(2) + \frac{1}{3} \cdot r(3)) + \frac{0.2}{6} \quad (5)$$

$$r(6) = 0.8(\frac{1}{6} \cdot r(1) + \frac{1}{5} \cdot r(2) + \frac{1}{3} \cdot r(3) + \frac{1}{2} \cdot r(4)) + \frac{0.2}{6} \quad (6)$$

Note 1

Computer Updates

- You must ensure that auto-updates are disabled on your computer prior to the online assessment.
- Special consideration will NOT be awarded on the grounds that your computer performed an update during an online assessment.

Note 2

If you upload the wrong document or wrong version of your exam

- Students are responsible for uploading the correct version of the correct document. Once uploaded, there will be no opportunity to replace or re-upload your exam papers AFTER the end of the exam.
- The documents submitted will be the documents that are marked. There is NO provision for students who upload incorrect or incomplete documents.
- **Therefore, you must check the work before you submit.**

Note 3

Communication during the exam

- Students are NOT permitted to communicate with other people during the exam (including the reading and submission periods).
- Attempts to communicate with other students will be considered to be serious academic misconduct.
- This includes communication in person, by email, text, message, telephone, or internet, i.e., do the work yourself

Sharing answers with others or posting them online

- Any attempts to collaborate or share your answers with others will be considered a very serious case of academic misconduct

Note 4

Checklist

- Be logged in at your computer and ready to go 20 minutes before the exam commences.
- Ensure your device has power, and the charger is plugged in.
- If applicable remind your roommates or family that you'll be taking an exam to avoid interruptions.

Note 5

- You can attempt the questions in any order, arrange your time wisely.
- Read all questions carefully
- Be fully prepared before the exam: don't forget to eat lunch, take break and relax yourself, no need to panic

Thank you!