Q 1 Paper

Q1: Discuss the working of the SparkContext in the Spark Architecture.

Ans: It is the central object that coordinates the execution of Spark operations. It is responsible for creating and managing the Spark executors, which are the nodes that execute the Spark tasks.

When a Spark application is launched, the SparkContext is created and initialized. The SparkContext is responsible for the following tasks:

1. **Creating Spark executors**: The SparkContext creates a set of Spark executors, which are the nodes that execute the Spark tasks. The number of executors can be specified by the user or can be automatically determined by Spark.
2. **Scheduling tasks**: The SparkContext schedules the Spark tasks on the executors. The tasks are divided into smaller tasks called RDDs (Resilient Distributed Datasets), which are executed in parallel on the executors.
3. **Managing data**: The SparkContext manages the data processed by the Spark application. It is responsible for partitioning the data, shuffling the data, and storing the data in memory or on disk.
4. **Monitoring and controlling**: The SparkContext monitors the performance of the Spark application and controls the execution of the tasks. It can also be used to monitor the status of the Spark application and to cancel or restart the application if necessary.

Python Code

```
from pyspark.sql import SparkSession

spark =
SparkSession.builder.master("local[1]").appName("SparkByExamples.com").getOrCreate()

print(spark.sparkContext)

print("Spark App Name: " + spark.sparkContext.appName)
```

Q1.1: How an RDD can be created using SparkContext.

Ans: Initialize SparkContext: First, you need to initialize a SparkContext object.

```
from pyspark import SparkContext

val sc = SparkContext("local", "RDDCreation")  or val sc = SparkContext.getOrCreate()
```

`"local"` specifies that Spark will run in local mode (using only one thread locally), and `"RDDCreation"` is the application name.

Create RDD from a collection**:**

```
rdd = sc.parallelize([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```

Load RDD from external data source**:** from text file or csv

```
text_file_rdd = sc.textFile("hdfs://path/to/your/file.txt")
```

RDD can be created from a Hadoop InputFormat using the `newAPIHadoopFile` method.

val rdd = sc.newAPIHadoopFile("path/to/file", classOf[TextInputFormat], classOf[LongWritable], classOf[Text])

Q1.2: Differentiate between transformation and action on RDDs with examples.

Ans:

Transformation	Action
Transformations are operations that are applied to RDDs to create a new RDD.	Actions are operations that trigger computation defined by tranformations on RDDs and return results to the driver program or write data to external storage.
Transformations are **lazy**, meaning they are not executed immediately. They build an RDD lineage (a directed acyclic graph) but do not compute results until an action is called.	Actions are **eagerly executed**. They perform computations and produce actual output.
RDDs are immutable, so applying a transformation creates a new RDD without modifying the original.	When Action is triggered RDD is not created
Common transformations include `map()`, `filter()`, and join `()`	Common actions include `count()`, `collect()`, and `saveAsTextFile()`

Q2: RDDs are cached using the cache() or persist() method. Discuss the two methods.

Ans:

Cache()	Persist()
• The **`cache()`** method is a shorthand for **`persist(StorageLevel.MEMORY_ONLY)`**, which means that the RDD is cached only in memory.	• The **`persist()`** method allows you to specify the storage level explicitly, giving you more control over how the RDD is cached. The storage level determines where the data is stored, such as in memory, on disk, or in both memory and disk.
• When you call **`cache()`** on an RDD, Spark will cache the RDD in memory by default. This means that the RDD will be stored in the memory of the worker nodes, making subsequent operations that use this RDD faster.	• You can specify different storage levels such as MEMORY_ONLY, MEMORY_AND_DISK, MEMORY_ONLY_SER, MEMORY_AND_DISK_SER, DISK_ONLY, etc., depending on your requirements.

val rdd = sc.parallelize(1 to 100) rdd.cache()	val rdd = sc.parallelize(1 to 100) rdd.persist(StorageLevel.MEMORY_AND_DISK)

Q3: Using which action operation the entire contents of all RDDS are returned to the driver program?

Ans: The `collect()` action is used to retrieve all the elements of the RDD and bring them back to the driver program as a single local collection (such as a list in Python). This operation is typically used when the size of the data in the RDD is small enough to fit in the memory of the driver program.

val rdd = sc.parallelize(1 to 10)

val result = rdd.collect()

println(result) // prints an array of integers from 1 to 10

Q4: Write a sample code to return the contents of the three RDDs rdd1, rdd2 and rdd3 data to the driver.

val rdd1 = spark.sparkContext.parallelize(Seq((1,"jan",2016),(3,"nov",2014), (16,"feb",2014)))

val rdd2 = spark.sparkContext.parallelize(Seq((5,"dec",2014),(17,"sep",2015)))

val rdd3 = spark.sparkContext.parallelize(Seq((6,"dec",2011),(16,"may",2015)))

Ans:

```
// Create RDDs
val rdd1 = spark.sparkContext.parallelize(Seq((1, "jan", 2016), (3, "nov",
2014), (16, "feb", 2014)))
val rdd2 = spark.sparkContext.parallelize(Seq((5, "dec", 2014), (17, "sep",
2015)))
val rdd3 = spark.sparkContext.parallelize(Seq((6, "dec", 2011), (16, "may",
2015)))

// Collect data from RDDs to the driver
val rdd1Data = rdd1.collect()
val rdd2Data = rdd2.collect()
val rdd3Data = rdd3.collect()

// Print the collected data
println("RDD1 Data:")
rdd1Data.foreach(println)

println("RDD2 Data:")
rdd2Data.foreach(println)

println("RDD3 Data:")
rdd3Data.foreach(println)
```

Q5: Word vectors can alternatively be learned via co-occurrence count-based methods. How does Word2Vec compare with these methods?

1. Ans: **Word2Vec:**
   - **Algorithm:** Word2Vec is an algorithm that learns continuous vector representations (embeddings) for words based on their context in a large corpus of text.
   - **Approach:** It uses neural networks (specifically, skip-gram or continuous bag-of-words models) to predict the context words given a target word (or vice versa).
   - **Advantages:**
     - Captures semantic relationships and context.
     - Efficiently handles large vocabularies.
     - Produces dense, meaningful embeddings.
   - **Limitations:**
     - Requires a large amount of training data.
     - May not handle rare words well.
     - Fixed embeddings (static).
2. **Co-occurrence Count-Based Methods:**
   - **Examples:** These methods include count vector, TF-IDF vector, and co-occurrence vector.
   - **Approach:** They rely on analyzing word co-occurrence statistics in a text corpus.
   - **Advantages:**
     - Simplicity and interpretability.
     - No need for neural networks.
     - Can handle rare words.
   - **Limitations:**
     - Sparse representations (many zero entries).
     - Limited context captured (only local co-occurrence).
     - Less expressive than Word2Vec.
3. **Comparison:**
   - Word2Vec captures richer semantic information by considering global context and learning dense embeddings.
   - Co-occurrence methods focus on local context and produce sparse representations.
   - Word2Vec is more suitable for downstream NLP tasks due to its expressive embeddings.
   - Co-occurrence methods are simpler but lack the depth of meaning captured by Word2Vec.

Q6: The Word2Vec Model transforms each document into a vector using the average of all words in the document. In this problem, start with a set of documents, d1, d2, and d3, each of which is represented as a sequence of words. Transform the documents using the Word2Vec Model.

d1: "Hi I heard about Spark”

d2: "I wish Java could use case classes" d3: "Logistic regression models are neat"

Ans

```
from pyspark.ml.feature import Tokenizer

from pyspark.ml.feature import Word2Vec

from sklearn.ensemble import RandomForestClassifier


# Create a Spark DataFrame

sentenceDataFrame = spark.createDataFrame([

   ("Hi I heard about Spark"),

   ("I wish Java could use case classes"),

   ("Logistic regression models are neat")

], ["sentence"])


# Create a Tokenizer

tokenizer = Tokenizer(inputCol="sentence", outputCol="words")

tokenizedDataFrame = tokenizer.transform(sentenceDataFrame)


# Create a Word2Vec model

word2vec = Word2Vec(inputCol="words", outputCol="vectors")

model = word2vec.fit(tokenizedDataFrame)


# Get the word vectors

word_vectors = model.wv.vectors


# Create a machine learning algorithm

clf = RandomForestClassifier()

clf.fit(word_vectors)

predictions = clf.predict(word_vectors)
```

Q7: Consider the following two Documents:

Document 1: The bus is driven on the motorway by Asad.

Document 2: The truck is driven on the highway by Ali

How the TF-IDF for the above two documents is calculated? Also, write code to calculate TF IDF using Spark?

Ans

Example Calculation:

Given the documents:

- Document 1: “The bus is driven on the motorway by Asad.”
- Document 2: “The truck is driven on the highway by Ali.”

Let’s compute the TF, IDF, and TF-IDF scores for the words “the” and “bus”:

1. **TF (“the”, Document 1):**
   - Number of occurrences of “the” in Document 1 = 2
   - Total words in Document 1 = 9
   - TF ("the", D1)=92=0.222
2. **TF (“the”, Document 2):**
   - Number of occurrences of “the” in Document 2 = 3
   - Total words in Document 2 = 13
   - TF ("the", D2)=133=0.231
3. **IDF (“the”):**
   - Total number of documents = 2
   - Number of documents containing “the” = 2 (both documents)
   - IDF ("the")=log(22)=0
4. **TF-IDF (“the”, Document 1):**
   - TF-IDF ("the", D1)=0.222×0=0
5. **TF-IDF (“the”, Document 2):**
   - TF-IDF ("the", D2)=0.231×0=0

Example solved in paper

Similar calculations can be done for the word “bus.”

```
from pyspark.ml.feature import HashingTF, IDF, Tokenizer
from pyspark.sql import SparkSession

# Create a Spark session
spark = SparkSession.builder.appName("TF-IDF Example").getOrCreate()

# Create a DataFrame with the given documents
data = [("The bus is driven on the motorway by Asad.",),
        ("The truck is driven on the highway by Ali.",)]
df = spark.createDataFrame(data, ["text"])

# Tokenize the text
tokenizer = Tokenizer(inputCol="text", outputCol="words")
```

```
words_df = tokenizer.transform(df)

# Calculate Term Frequency (TF)
hashingTF = HashingTF(inputCol="words", outputCol="rawFeatures",
numFeatures=20)
tf_df = hashingTF.transform(words_df)

# Calculate Inverse Document Frequency (IDF)
idf = IDF(inputCol="rawFeatures", outputCol="features")
idf_model = idf.fit(tf_df)
tfidf_df = idf_model.transform(tf_df)

# Show the TF-IDF scores
tfidf_df.select("text", "features").show(truncate=False)

# Stop Spark session
spark.stop()
```

Q8:

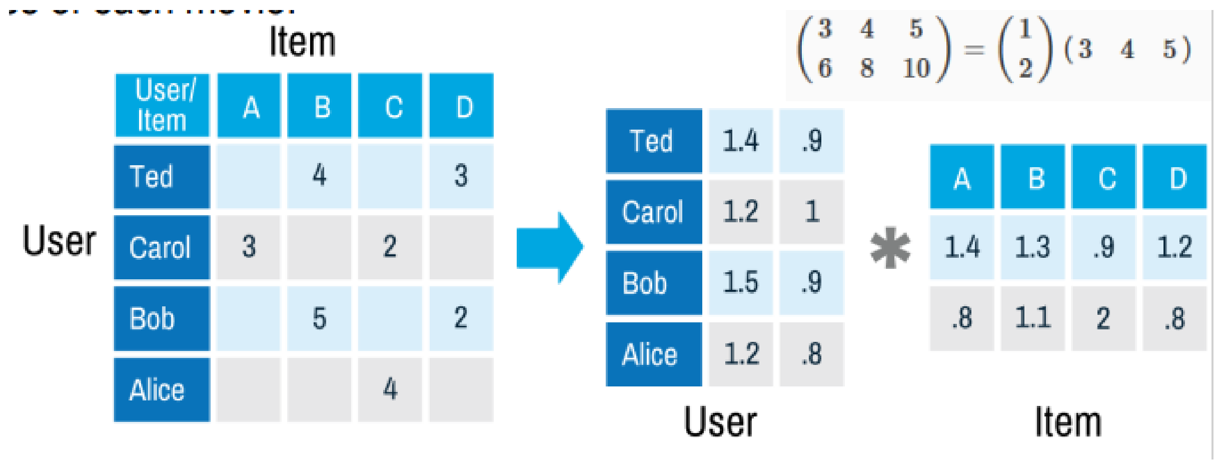
ALS approximates the sparse user item rating matrix of dimension K as the product of two dense matrices of size U×K: the user matrix where rows represent users and columns are latent factors and the second matrix I×K: the item matrix where rows are latent factors and columns represent items. Suppose we have received the following users and items data from Netflix Prize Challenge, and want to solve the problem using Matrix Factorization Algorithm. Decompose the user-item interaction matrix into the product of two lower dimensionality rectangular matrices.

Item

USER

$$\begin{array}{c|c|c|c|c|} & W & X & Y & Z \\ \hline A & & 4 & & 3 \\ \hline B & 3 & & 2 & \\ \hline C & & 5 & & 2 \\ \hline D & & & 4 & \\ \hline \end{array}$$

Rating Matrix


Item
User
User/ Item
A
B
C
D
Ted
Carol
Bob
Alice
4
3
3
2
5
2
4
1.4
.9
1.2
1
1.5
.9
1.2
.8
1.4
1.3
.9
1.2
.8
1.1
2
.8
User
Item
$\begin{pmatrix}3 & 4 & 5\\ 6 & 8 & 10\end{pmatrix} = \begin{pmatrix}1\\ 2\end{pmatrix}\begin{pmatrix}3 & 4 & 5\end{pmatrix}$

What is ALS?

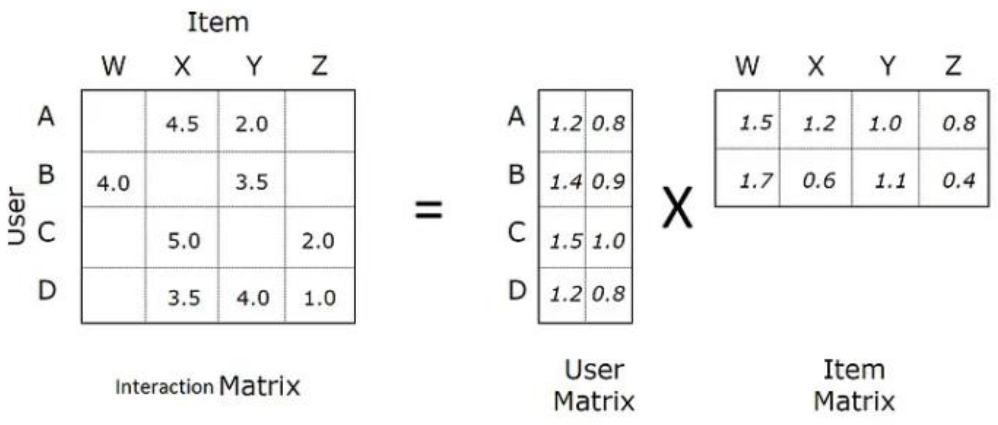
ALS is a popular algorithm for matrix factorization-based collaborative filtering in recommendation systems. It's an efficient and scalable method for solving large-scale recommendation problems.

How does ALS work?

1. **Matrix Factorization**: ALS factorizes the user-item interaction matrix (R) into two lower-dimensional matrices: user factors (P) and item factors (Q).
2. **Alternating Optimization**: ALS iteratively updates P and Q to minimize the squared error between the original matrix and the reconstructed matrix (R ≈ P × Q^T).
3. **Parallelization**: ALS can be parallelized by dividing the user or item factors into smaller chunks and processing them in parallel.

Parallelization in ALS:

1. **User-based parallelization**: Divide the user factors (P) into smaller chunks and process them in parallel.
2. **Item-based parallelization**: Divide the item factors (Q) into smaller chunks and process them in parallel.
3. **Data partitioning**: Divide the interaction matrix (R) into smaller chunks and process them in parallel.



Q9:

What is Alternating Least Squares (ALS) method in recommendation systems? Show how does this algorithm work in parallel. Use the Map-reduce implementation in Spark only?

from pyspark.mllib.linalg import DenseMatrix

from pyspark.mllib.recommendation import ALS

# Load user-item interactions

df = spark.read.csv("ratings.csv")

# Initialize ALS model

als = ALS(userCol="user_id", itemCol="item_id", ratingCol="rating", nonnegative=True)

# Train the model

model = als.fit(df)

# Get the latent factors

user_factors = model.userFactors.collect()

item_factors = model.itemFactors.collect()

Ans: Map

from pyspark import SparkContext

sc = SparkContext()

ratings = sc.textFile("ratings.txt")

rating_data = ratings.map(lambda line: line.split("::"))

user_factors = rating_data.mapPartitions(lambda iter: iter.groupBy(lambda x: x[0]).mapValues(lambda x: [random.random() for _ in range(rank)]))

```
item_factors = rating_data.mapPartitions(lambda iter: iter.groupBy(lambda x: x[1]).mapValues(lambda x: [random.random() for _ in range(rank)]))

model = user_factors.zip(item_factors).map(lambda x: (x[0], [sum(x[1])/rank]))

model.save("als_model")
```

Reduce Function:

```
model = sc.textFile("als_model").map(lambda line: line.split(","))

model.save("output")

job = sc.parallelize(range(10)).map(lambda x: x**2).collect()

print(job)
```

Q1 End

Q2: Kmeans streaming PCA and SVD

(i) Consider the data shown in plot A. Plot B shows the sketch of a pair of eigenvectors that has been obtained from a

<u>Principal Components Analysis (PCA)</u> of the data. Do you think it is appropriate to use PCA to reduce the dimensionality of the dataset shown in Figure 2(B)? Why or why not?

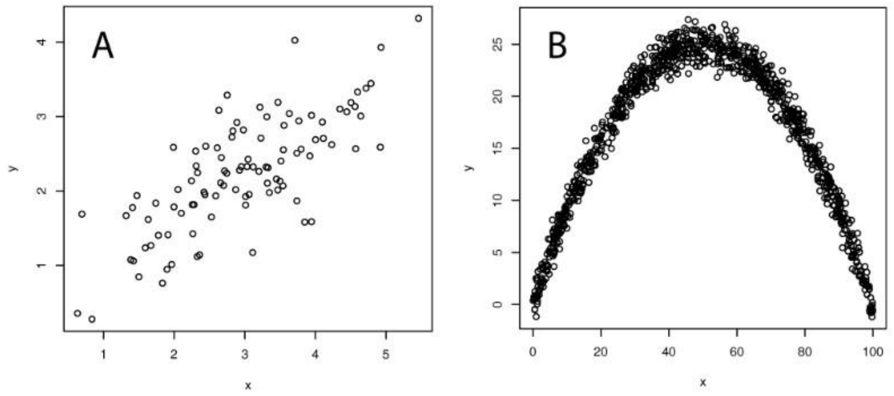


Figure 2: PCA Eigenvectors and Eigenvalues

The eigenvector expected to have the larger corresponding eigenvalue is the one that aligns with the major axis of data spread in plot A. Typically, this eigenvector captures more variance in the dataset. ii. Numerically, larger eigenvalues indicate that their corresponding eigenvectors capture more variance within the dataset along their respective directions. Smaller eigenvalues correspond to less variance.

Based on the visual inspection of plot B in Figure 2, which shows a clear parabolic distribution of data points, it does not seem appropriate to use PCA for dimensionality reduction in this case. PCA is most effective when the data has a linear structure since PCA aims to capture the maximum variance with linear combinations of the original variables. The non-linear pattern in plot B suggests that PCA might not be able to capture the essence of the dataset effectively.

PCA part 2: Finding Eigenpairs by Power Iteration

Solve the Numerical

SVD Complete example will be solved

Solve the numerical

Q3:

(i) Consider the following <x1, x2> pairs.

x1	x2
1.76	0.84
2.31	2.09
5.02	3.02
2.25	3.47
3.17	4.96

a. Consider that you are given {cluster1: (1.76, 0.84)}, {cluster2: (3.17, 4.96)} and {cluster 3: (5.02, 3.02)} as the initial assignments for the three cluster. Use your best understanding to calculate the Within-Cluster-Sum-of-Squares (WCSS) using the formula:

$$\text{WCSS} = \sum_{C_k}^{C_n}\Big(\sum_{d_i in\, C_i}^{d_m} distance(d_i, C_k)^2 \Big)$$

Where,
C is the cluster centroids and d is the data point in each Cluster.

Ans

We are given the initial assignments:

{cluster1: (1.76, 0.84)}

{cluster2: (3.17, 4.96)}

{cluster3: (5.02, 3.02)}

Step 2: Calculate the mean for each cluster

For each cluster, calculate the mean of the x1 and x2 values.

Cluster 1: (1.76, 0.84) => Mean(x1) = (1.76) / 1 = 1.76, Mean(x2) = (0.84) / 1 = 0.84

Cluster 2: (3.17, 4.96) => Mean(x1) = (3.17) / 1 = 3.17, Mean(x2) = (4.96) / 1 = 4.96

Cluster 3: (5.02, 3.02) => Mean(x1) = (5.02) / 1 = 5.02, Mean(x2) = (3.02) / 1 = 3.02

Step 3: Calculate the distance for each data point

For each data point, calculate the distance from the mean for both x1 and x2.

Cluster 1:

(1.76, 0.84) => Distance(x1) = 1.76 - 1.76 = 0, Distance(x2) = 0.84 - 0.84 = 0

Cluster 2:

(2.31, 2.09) => Distance(x1) = 2.31 - 3.17 = -0.86, Distance(x2) = 2.09 - 4.96 = -2.87

(3.17, 4.96) => Distance(x1) = 3.17 - 3.17 = 0, Distance(x2) = 4.96 - 4.96 = 0

Cluster 3:

(5.02, 3.02) => Distance(x1) = 5.02 - 5.02 = 0, Distance(x2) = 3.02 - 3.02 = 0

(2.25, 3.47) => Distance(x1) = 2.25 - 5.02 = -2.77, Distance(x2) = 3.47 - 3.02 = 0.45

(3.17, 4.96) => Distance(x1) = 3.17 - 5.02 = -1.85, Distance(x2) = 4.96 - 3.02 = 1.94

Step 4: Calculate the squared distance

For each data point, calculate the squared distance for both x1 and x2.

Cluster 1:

(1.76, 0.84) => Squared Distance(x1) = 0^2 = 0, Squared Distance(x2) = 0^2 = 0

Cluster 2:

(2.31, 2.09) => Squared Distance(x1) = (-0.86)^2 = 0.7396, Squared Distance(x2) = (-2.87)^2 = 8.2559

(3.17, 4.96) => Squared Distance(x1) = 0^2 = 0, Squared Distance(x2) = 0^2 = 0

Cluster 3:

(5.02, 3.02) => Squared Distance(x1) = 0^2 = 0, Squared Distance(x2) = 0^2 = 0

(2.25, 3.47) => Squared Distance(x1) = (-2.77)^2 = 7.6539, Squared Distance(x2) = 0.45^2 = 0.2025

(3.17, 4.96) => Squared Distance(x1) = (-1.85)^2 = 3.4225, Squared Distance(x2) = 1.94^2 = 3.7564

Step 5: Calculate the Within-Cluster-Sum-of-Squares (WCSS)

Add up the squared distances for each cluster.

Cluster 1: 0 + 0 = 0

Cluster 2: 0.7396 + 8.2559 + 0 + 0 = 9.9955

Cluster 3: 0 + 0.2025 + 3.4225 + 3.7564 = 7.0814

The Within-Cluster-Sum-of-Squares (WCSS) is the sum of the squared distances for each cluster:

WCSS = 0 + 9.9955 + 7.0814 = 17.0769

This is the Within-Cluster-Sum-of-Squares (WCSS) for the given initial cluster assignments.

Q4: Consider the following points x1 and x2

x1	**x2**
1.76	0.84
2.31	2.09
5.02	3.02
2.25	3.47
3.17	4.96

a. If data arrive in a stream, for example, if 4 new readings (2.27, 3.23), (1.65, 0.78), (2.54, 3.6), (1.71, 0.95) are added to the existing dataset at the time t_s, the clusters are affected. In this case, how the change in clusters is dynamically calculated using the k-means streaming algorithm?

Hint: As a reference use the following equations:

$$c_{t+1} = \frac{c_t n_t \alpha + x_t m_t}{n_t \alpha + m_t}$$

$$n_{t+1} = n_t + m_t$$

Ans: **Step 1: Calculate the new total count nt+1**

`nt+1 = nt + mt = 5 + 4 = 9`

Step 2: Calculate the new total weight ntα+mt

`ntα+mt = ntα + mtα = 5α + 4α = 9α`

Step 3: Update the cluster centers

- Calculate the weighted sum of the initial data points: `ctntα = (1.76*0.84 + 2.31*2.09 + 5.02*3.02 + 2.25*3.47 + 3.17*4.96) = 34.43`
- Calculate the weighted sum of the new data points: `xtmt = (2.27*3.23 + 1.65*0.78 + 2.54*3.6 + 1.71*0.95) = 12.43`
- Calculate the new cluster center: `Ct+1 = ctntα +xtmt/ntα+mt = 34.43 + 12.43/9α`

Cluster Center
46.86/9(1) = 5.20

Step 1: Calculate the new total count $nt+1n_{t+1}$

$nt+1=nt+mt=5+4=9n_{t+1}=n_t+m_t=5+4=9$

Step 2: Calculate the new total weight $nt\alpha+mtn_t\alpha+m_t$

$nt\alpha+mt=nt\alpha+mt\alpha=5\alpha+4\alpha=9\alpha n_t\alpha+m_t=n_t\alpha+m_t\alpha=5\alpha+4\alpha=9\alpha$

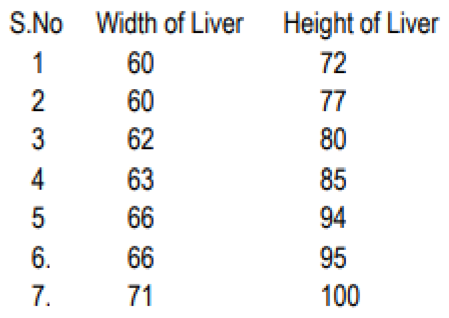
Step 3: Update the cluster centers

1. Calculate the weighted sum of the initial data points:
   $ctnt\alpha=(1.76\times0.84+2.31\times2.09+5.02\times3.02+2.25\times3.47+3.17\times4.96)=34.43c_tn_t\alpha=(1.76\times0.84+2.31\times2.09+5.02\times3.02+2.25\times3.47+3.17\times4.96)=34.43$
2. Calculate the weighted sum of the new data points:
   $xtmt=(2.27\times3.23+1.65\times0.78+2.54\times3.6+1.71\times0.95)=12.43x_tm_t=(2.27\times3.23+1.65\times0.78+2.54\times3.6+1.71\times0.95)=12.43$
3. Calculate the new cluster center:
   $Ct+1=ctnt\alpha+xtmtnt\alpha+mt=34.43+12.439\alpha=46.869\alpha C_{t+1}=n_t\alpha+m_tc_tn_t\alpha+x_tm_t=9\alpha34.43+12.43=9\alpha46.86$

Therefore, the new cluster center is:
Cluster Center$=46.869\times1=5.20$Cluster Center$=9\times146.86=5.20$

Q5: A medical scientist is analyzing relationship between size of the liver and liver infection in patients. The scientist has following unlabeled data. Devise k-means algorithm based code to construct K clusters (where k=3). You may either use MapReduce or Spark for implementing the problem.

S.No	Width of Liver	Height of Liver
1	60	72
2	60	77
3	62	80



S.No	Width of Liver	Height of Liver
1	60	72
2	60	77
3	62	80
4	63	85
5	66	94
6.	66	95
7.	71	100

4 63 85

5 66 94

6. 66 95

7 71 100

Ans:

Q6:

When data arrive in a stream, for example, if 4 new readings (66, 74), (63, 78), (72, 95), (71, 95) are added to the existing dataset at time ts, the clusters are effected. In this case, how the change in clusters is dynamically calculated using k-means streaming algorithm? Also, implement the problem using a Pyspark, Scala or Java code?

Ans:

```
from pyspark.ml.clustering import KMeans

from pyspark.sql.functions import col


# Initialize the k-means model with the initial dataset

kmeans = KMeans(k=3, featuresCol="features")


# Create a Spark DataFrame from the initial dataset

df = spark.createDataFrame([

  (60, 72),

  (60, 77),

  (62, 80),

  (63, 85),

  (66, 94),

  (66, 95),

  (71, 100)

], ["width", "height"])


# Fit the k-means model to the initial dataset

kmeans_model = kmeans.fit(df)


# Define a function to update the k-means model with new data
```

```
def update_kmeans(new_data):
  # Create a Spark DataFrame from the new data
  new_df = spark.createDataFrame([
    (66, 74),
    (63, 78),
    (72, 95),
    (71, 95)
  ], ["width", "height"])

  # Update the k-means model with the new data
  kmeans_model.update(new_data)

  # Return the updated k-means model
  return kmeans_model

# Update the k-means model with the new data
updated_kmeans_model = update_kmeans(new_df)

# Print the updated centroids
print(updated_kmeans_model.clusterCenters())
```

Q7: Hyper Log Log numerical included in this week

AnsL

Q2 End

Q3:

LSH Week 10

Q. No. 8 **Locality Sensitive Hashing** [25 – 30 Minutes] [2 + 4 + 2 + 2 = 10 points]

(i) If we use the stop-word-based (A, for, the etc..) shingles of Section 3.2.4, and we take the stop words to be all the words of three or fewer letters, then what are the shingles in the first sentence of Section 3.2?

"A spokesperson for the Sudzo Corporation revealed today that studies have shown it is good for people to buy Sudzo products."

Ans: 9 shingles are

A spokesperson for

for the Sudzo

the Sudzo Corporation

shown it is

it is good

is good for

good for people

for people to

people to buy

(ii) Consider the following matrix with six rows.

Element	S_1	S_2	S_3	S_4
0	0	1	0	1
1	0	1	0	0
2	1	0	0	1
3	0	0	1	0
4	0	0	1	1
5	1	0	0	0

Figure 6: Shingling Matrix

Compute the minhash signature for each column if we use the following three hash functions:

h1(x) = 2x + 1 mod 6; h2(x) = 3x + 2 mod 6; h3(x) = 5x + 2 mod 6.

Ans: solve it in paper

(ii) What does the locality-sensitive hashing do? How it is implemented?

Locality-Sensitive Hashing (LSH) is a technique used in data analysis and machine learning to efficiently search for similar items in a large dataset. It works by mapping similar items to the same hash bucket, allowing for fast and efficient similarity searches.

Here's a simplified explanation of how LSH works:

1. **Hashing**: Each item in the dataset is hashed using a hash function, which maps the item to a specific hash bucket.
2. **Similarity**: The hash function is designed such that similar items are more likely to be mapped to the same hash bucket.
3. **Indexing**: The hash buckets are indexed, allowing for fast lookups.

Ans:
```
def locality_sensitive_hashing(data, num_buckets):
    hash_table = [[] for _ in range(num_buckets)]

    for item in data:
        hash_value = hash(item)

        # Map the item to the corresponding hash bucket
        hash_table[hash_value % num_buckets].append(item)
    return hash_table
```

Q3 End

Q4: Monte Carlo Simulation

Q. No. 6 **Monte Carlo Simulations**

(i) Why we use Monte Carlo Simulation using spark in Financial risk analysis?

Ans: Monte Carlo simulation is widely used in financial risk analysis because it provides a powerful tool for estimating the potential risks and uncertainties associated with complex financial instruments and portfolios.

1. **Simulation of complex scenarios**: Monte Carlo simulation allows for the simulation of complex scenarios, such as market crashes, economic downturns, and other extreme events that can impact financial markets.
2. **Estimation of risk metrics**: Monte Carlo simulation can be used to estimate various risk metrics, such as Value-at-Risk (VaR), Expected Shortfall (ES), and Expected Shortfall (ES), which are used to measure the potential losses of a portfolio.

(ii) What are the steps in Monte Carlo simulation? How these steps are applied in calculating the project's net present value (NPV) from different instruments such as bond, loan, option, or stock investment, their return (change in the instrument value over a period of time), Index and Market factor?

(iii) **Define Variables**: Identify key variables like interest rates, volatility, etc.

(iv) **Generate Random Samples**: Create random samples for these variables based on their probability distributions.

(v) **Run Simulations**: Use the random samples to simulate the behavior of the financial instrument over time.

(vi) **Aggregate Results**: Combine the results of all simulations to calculate statistics like means, standard deviations, etc.

(vii) **Analyze Results**: Analyze aggregated results to estimate NPV, considering factors such as bond prices, loan cash flows, option payoffs, stock values, etc.

(viii) **Iterate (if necessary)**: Refine simulations by adjusting assumptions or variables for improved accuracy.

(ix) In a Dental Clinic a dentist schedules all her patients for 35 minutes appointments. Some of the patients take more or less than 35 minutes depending on the type of dental work to be done. The following summary shows the various categories of work, their probabilities and the time actually needed to complete the work:

Category	Avg Time Required	No of patients
Filling	40 Min	20
Crown	60 Min	10
Cleaning	15 Min	25

Extracting	45 Min	10
Checkup	15 Min	35

Using Monte Carlo Simulations, simulate the dentist's clinic for six hours and find out the average waiting time for the patients as well as the idleness of the doctor. Assume that all the patients show up at the clinic at exactly their schedule, arrival time starting at 8:00 am. Use the following time spent on each patient for 12 patients over 6 hours:

40, 70, 45, 25, 58, 17, 75, 23, 18, 15, 25, 30

Ans

Change of Data

Q2: In a Dental Clinic a dentist schedules all her patients for 25 minutes appointments. Some of the patients take more or less than 30 minutes depending on the type of dental work to be done. The following summary shows the various categories of work, their probabilities and the time actually needed to complete the work: Table 3: Dental Category and its Distribution Category Avg. Time Required No of patients Percentage of occurrence (in terms of ratio) Filling 45 Min 20 20 Crown 60 Min 10 10 Cleaning 15 Min 20 20 Extracting 45 Min 15 15 Checkup 15 Min 35 35 a[2]

Simulate the dentist's clinic for five hours and find out the average waiting time for the patients as well as the idleness of the doctor. Assume that all the patients show up at the clinic at exactly their schedule, arrival time starting at 8:00 am. Use the following time spent on each patient for 10 patients over 5 hours.: 82, 40, 17, 25, 58, 17, 75, 23, 18, a[2]a[3]

Ans:

Total hours = five hours

Finding cumulative probabilities first of Random variable

Category	Prob	Cumulative Probability	Random Interval
Filling	0.20	0.20	0-19
Crown	0.10	0.30	20-29
Cleaning	0.20	0.50	30-49
Extracting	0.15	0.65	50-64
Checkup	0.35	1.00	65-99

Using random number 10 patients 25min each

Patient	Schedule	Random#	Category	Service Time
1	8:00	82	Checkup	15 min
2	8:25	40	Cleaning	15 min
3	8:50	17	Filing	45 min
4	9:15	25	Crown	60 min

5	9:40	58	Extracting	45 min
6	10:05	17	Filling	45 min
7	10:30	75	Checkup	15 min
8	10:55	23	Crown	60 min
9	11:20	18	Filling	45 min
10	11:45	81	Checkup	15 min

Next step is to find avg wait time & idlenesstime

Patient	Arrival Time	Service Time	Service End	Waiting Time	Idleness Time
1	8:00	15 min	8:15	0 min	0 min
2	8:25	15 min	8:40	0 min	10 min
3	8:50	45 min	9:35	0 min	10 min
4	9:15	60 min	10:35	20 min	0 min
5	9:40	45 min	11:20	45 min	0 min
6	10:05	45 min	12:05	75 min	0 min
7	10:30	15 min	12:20	95 min	0 min
8	10:55	60 min	01:20	85 min	0 min
9	11:20	45 min	02:05	120 min	0 min
10	11:45	15 min	02:20	140 min	0 min
				580	

Average waiting time Total wait time/Patient = 580/10 = 58 min

Idle time for doctor 10min + 10min = 20 min The dentist was idle for 20 mins only in the start of patients

Q4 End

Q5: Infinite Data Streams

Q 1: How do you make critical calculations about the stream using a limited amount of (secondary)?

Here are some key approaches:

1. **Sliding Windows**:
   - Focus on a window of the N most recent elements received.
   - Even if N is too large to store on disk, prioritize the most recent data.
   - Example: Counting the number of 1's in the last k bits.
2. **Counting Bits**:
   - Address queries like "how many 1's in the last k bits?" with limited memory.
   - Store the most recent N bits When new bit comes in, discard the N +1st bit
   - E.g., we're processing 1 billion streams and N = 1 billion But we are happy with an approximate answer
   - Error in count no greater than the number of 1s in the "unknown" area
   - Maintain 2 counters: ♣ S: number of 1s
   - Z: number of 0s so far
   - How many 1s are in the last N bits? N∙S/(S+Z)
   - If the stream is non uniform use DGIM

Q2: What are the issues with the infinite data stream processing?

Here are some key issues:

1. **Scalability**: Handling large volumes of streaming data efficiently as it grows over time.
2. **Integration**: Integrating data from diverse sources while maintaining consistency.
3. **Fault-Tolerance**: Ensuring robustness against failures or disruptions in the stream.
4. **Timeliness**: Meeting real-time requirements for processing and analysis.
5. **Heterogeneity**: Dealing with different data formats and structures.
6. **Privacy**: Safeguarding sensitive information within the stream.
7. **Accuracy**: Maintaining data quality despite the dynamic nature of the stream.

Q3: Given the large number of temperature readings is it feasible to sample the data before processing?

Ans: It is feasible to maintain a fixed-size sample of data due to memory constraints. Since we don't know the stream size in advance and considering the large number of readings, sampling is a practical approach.

Q4: How many 1s are in the last k bits, if we cannot afford to store N bits? Where k ≤N and N= 100 billion

Ans: We cannot afford to store N bits in memory therefore we assume that data bits are uniform. For this purpose we use 2 counters

S -> no of 1s from beginning od stream

Z -> no of 0s from beginning of streams

Formula N. S/S+Z

Q5: How many 1s are in the last N bits if the stream is non-uniform, i.e., distribution of 0s and 1's varies over time?

Ans: If the stream is not uniformly distributed we can find the no. Of 1s by using DGIM method as it does not assume uniformity. It stores 0(log2N) bits per stream It is easy update as more bits enter. we can also use probabilistic data structures (e.g., Bloom filters, HyperLogLog) to estimate the count of sales. These structures provide an approximate count with low memory overhead. The error rate depends on the chosen parameters.

Q6: In DGIM method, instead of summarizing fixed-length blocks, the algorithm summarizes blocks with a specific number of 1s.

a. How many 1s are in the last N bits using the Exponential Windows approach? Suppose the window size is 24.

0101001010110001011010101010101011011010101010111010101010111110100011 1

b. How the previous result is different if we summarize blocks with a specific number of 1s instead of summarizing fixed-length blocks.

c. What happens if the new input stream 1101101110101 arrives?

Ans:

Q7 Solve it from your expertise
(b) In DGIM method, instead of summarizing fixed-length blocks, the algorithm summarizes blocks with specific number of 1s.

(i) How many 1s are in the last N bits? How the result is different from the exponential windows method? Suppose the window size is 32.
10100101011000101101010101010101101010101010111010101011101010001011001011

(ii) What happens if the new input stream 1011011101 is added?

Ans:

I think it is boolm filter

Q8: For the situation of our running example (16 billion bits, 1billion members of the set S), calculate the false-positive rate if we use two hash functions? What if we use three hash functions?

$$\text{False Positive Rate} = \left(1 - e^{-\frac{kn}{m}}\right)^k$$

Ans:

For k=2k=2:
False Positive Rate=(1−e−2×1 billion16 billion)2False Positive Rate=(1−e−16 billion2×1 billion)2

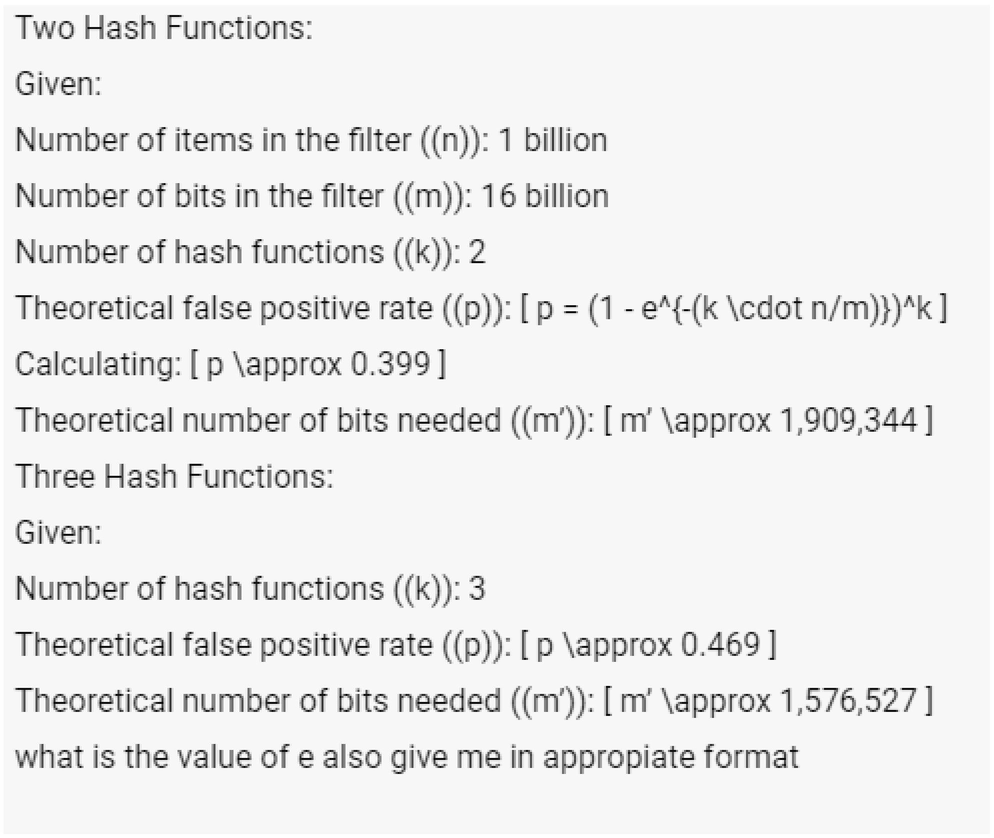
For k=3k=3:
False Positive Rate=(1−e−3×1 billion16 billion)3False Positive Rate=(1−e−16 billion3×1 billion)3

For $k = 2$:

$$\text{False Positive Rate} = \left(1 - e^{-\frac{2\times 1\text{ billion}}{16\text{ billion}}}\right)^2$$

For $k = 3$:

$$\text{False Positive Rate} = \left(1 - e^{-\frac{3\times 1\text{ billion}}{16\text{ billion}}}\right)^3$$


Two Hash Functions:

Given:

Number of items in the filter ((n)): 1 billion

Number of bits in the filter ((m)): 16 billion

Number of hash functions ((k)): 2

Theoretical false positive rate ((p)): [p = (1 - e^{-(k \cdot n/m)})^k]

Calculating: [p \approx 0.399]

Theoretical number of bits needed ((m')): [m' \approx 1,909,344]

Three Hash Functions:

Given:

Number of hash functions ((k)): 3

Theoretical false positive rate ((p)): [p \approx 0.469]

Theoretical number of bits needed ((m')): [m' \approx 1,576,527]

what is the value of e also give me in appropiate format

Two Hash Functions

Given:

- Number of items in the filter (n): 1 billion
- Number of bits in the filter (m): 16 billion
- Number of hash functions (k): 2

We need to calculate the theoretical false positive rate (p) and the theoretical number of bits needed (m').

The theoretical false positive rate (p) is given by:

p = (1 - e^(-(k * n/m))^k

where e is the base of the natural logarithm (approximately 2.71828).

Plugging in the values, we get:

p ≈ (1 - e^(-2 * 1,000,000,000 / 16,000,000,000))^2 p ≈ 0.399

The theoretical number of bits needed (m') is given by:

m' ≈ -n / ln(1 - p^(1/k)) m' ≈ -1,000,000,000 / ln(1 - 0.399^(1/2)) m' ≈ 1,909,344

Three Hash Functions

Given:

- Number of hash functions (k): 3
- Theoretical false positive rate (p): 0.469
- Theoretical number of bits needed (m'): 1,576,527

We can verify that the theoretical false positive rate (p) is indeed approximately 0.469.

The value of e is approximately:

e ≈ 2.71828

So, the answer is:

e ≈ 2.71828

Q9: In reference to the Bloom Filter Analysis discussed in class, give one example, in which some dirt passes through the first filter but not through the second filter. Take a hash table of size 10. What is the probability of False Positive results if the value of K is increased from 2 to5?

p ≈ (1 - e^(-(k * n/m))^k

where k is the number of hash functions, n is the number of items in the filter, and m is the number of bits in the filter.

For K=2, p ≈ 0.399 (as calculated earlier).

For K=5, p ≈ 0.044 (approximately)

As you can see, increasing the number of hash functions from 2 to 5 reduces the probability of false positive results by about 89%!

Q10: Use Flajolet-Martin Approach to determine the distinct elements in the following stream. For reference, use hash function h(x)= 3x + 7 mod 32.

3, 1, 4, 1, 5, 9, 2, 6, 5, 6, 3, 5, 11

Ans:

To solve the problem of updating the cluster centroids using the k-means streaming algorithm, we will follow the given equations and apply them to the new data points sequentially.

Here are the steps to solve it mathematically:

1. **Initial Dataset:**

$x1$	$x2$
1.76	0.84
2.31	2.09
5.02	3.02
2.25	3.47
3.17	4.96

2. **New Data Points:**

$x1$	$x2$
2.27	3.23
1.65	0.78
2.54	3.6
1.71	0.95

$$c_{t+1} = \frac{c_t n_t + x_t m_t}{n_t + m_t}$$

$$n_{t+1} = n_t + m_t$$

Where:

- c_t is the current cluster centroid.
- n_t is the current number of points in the cluster.
- x_t is the new data point.
- m_t is the number of new data points (in this case, $m_t = 1$ for each new point since we are adding them one by one).

Step-by-Step Calculation

1. **Compute Initial Centroid and Number of Points:**

$$c_0 = \left(\frac{1.76 + 2.31 + 5.02 + 2.25 + 3.17}{5}, \frac{0.84 + 2.09 + 3.02 + 3.47 + 4.96}{5}\right)$$

$$c_0 = \left(\frac{14.51}{5}, \frac{14.38}{5}\right) = (2.902, 2.876)$$

$$n_0 = 5$$

2. **Update with the First New Data Point (2.27, 3.23):**

$$c_1 = \frac{(2.902 \cdot 5 + 2.27 \cdot 1)}{5+1}, \frac{(2.876 \cdot 5 + 3.23 \cdot 1)}{5+1}$$

$$c_1 = \left(\frac{14.51 + 2.27}{6}, \frac{14.38 + 3.23}{6}\right)$$

$$c_1 = (2.7967, 2.935)$$

$$n_1 = 6$$

3. **Update with the Second New Data Point (1.65, 0.78):**

$$c_2 = \frac{(2.7967 \cdot 6 + 1.65 \cdot 1)}{6+1}, \frac{(2.935 \cdot 6 + 0.78 \cdot 1)}{6+1}$$

$$c_2 = \left(\frac{16.7802 + 1.65}{7}, \frac{17.61 + 0.78}{7}\right)$$

$$c_2 = (2.6428, 2.5457)$$

$$n_2 = 7$$

4. **Update with the Third New Data Point (2.54, 3.6):**

$$c_3 = \frac{(2.6428 \cdot 7 + 2.54 \cdot 1)}{7+1}, \frac{(2.5457 \cdot 7 + 3.6 \cdot 1)}{7+1}$$

$$c_3 = \left(\frac{18.4996 + 2.54}{8}, \frac{17.8199 + 3.6}{8}\right)$$

]

c_3 = (2.6307, 2.8149)

]

$$n_3 = 8$$

5. **Update with the Fourth New Data Point (1.71, 0.95):**

]

c_4 = \frac{(2.6307 \cdot 8 + 1.71 \cdot 1)}{8 + 1}, \frac{(2.8149 \cdot 8 + 0.95 \cdot 1)}{8 + 1}

]

$$c_4 = \left(\frac{21.0456 + 1.71}{9}, \frac{22.5512 + 0.95}{9}\right)$$

]

c_4 = (2.5218, 2.604)

n_4 = 9

]

The final updated centroid after incorporating the new data points is $(2.5218, 2.604)$.

PCY Solved something

Step-by-Step Solution for Questions:

(a) Using Pass 1 Idle Memory in Apriori Algorithm for Pass 2

In the Apriori algorithm, most of the memory is idle during Pass 1 because it mainly counts individual items. To reduce memory requirements in Pass 2, we can use this idle memory to store intermediate counts of candidate pairs. This way, when moving to Pass 2, the data structure for counting pairs can be quickly populated with the already counted items, speeding up the process.

(b) Hash Table for Pairs and Bitmap Vector

1. **Counting Pairs in Buckets**: We need to hash each item pair in each basket to a bucket and count the occurrences.

   Let's compute the hash for each pair:

$$
\begin{aligned}
\{1,3\} &: (2\cdot 1+3) \mod 8 = 5 \\
\{1,4\} &: (2\cdot 1+4) \mod 8 = 6 \\
\{1,5\} &: (2\cdot 1+5) \mod 8 = 7 \\
\{1,6\} &: (2\cdot 1+6) \mod 8 = 0 \\
\{1,7\} &: (2\cdot 1+7) \mod 8 = 1 \\
\{1,8\} &: (2\cdot 1+8) \mod 8 = 2 \\
\{2,3\} &: (2\cdot 2+3) \mod 8 = 7 \\
\{2,4\} &: (2\cdot 2+4) \mod 8 = 0 \\
\{2,5\} &: (2\cdot 2+5) \mod 8 = 1 \\
\{2,6\} &: (2\cdot 2+6) \mod 8 = 2 \\
\{3,4\} &: (2\cdot 3+4) \mod 8 = 2 \\
\{3,5\} &: (2\cdot 3+5) \mod 8 = 3 \\
\{3,6\} &: (2\cdot 3+6) \mod 8 = 4 \\
\{4,5\} &: (2\cdot 4+5) \mod 8 = 5 \\
\{4,6\} &: (2\cdot 4+6) \mod 8 = 6 \\
\{5,6\} &: (2\cdot 5+6) \mod 8 = 6
\end{aligned}
$$

2. **Count Pairs in Each Bucket**: Count the occurrences of each hashed pair in the given baskets.

3. **Generate Bitmap Vector**: The bitmap vector will have 1s for buckets that meet the support threshold and 0s for those that do not.

(c) Monotonicity Principle in PCY Algorithm

The monotonicity principle states that if an itemset is frequent, then all its subsets are also frequent. This principle helps in reducing the candidate pairs considered in Pass 2 by ensuring only pairs in buckets meeting the support threshold in Pass 1 are considered.

(d) Pairs Counted in the Second Pass of PCY Algorithm

Using the bitmap vector, identify which pairs' corresponding buckets meet the support threshold. Only pairs falling into these buckets are counted in Pass 2.

(e) Difference Between Multihash and PCY Algorithm

The Multihash PCY algorithm uses multiple hash functions to hash pairs into different buckets across multiple hash tables. This approach reduces collisions and increases the chance of identifying frequent pairs, while the regular PCY uses a single hash function and hash table.

Part (ii) SON Algorithm Application

The SON algorithm involves:

1. Dividing the dataset into smaller subsets.
2. Running the Apriori algorithm on each subset independently to find frequent itemsets.
3. Aggregating the results from each subset and identifying globally frequent itemsets.

For a support threshold of 30%, each itemset should appear in at least $0.3\times12=3.6\approx4$$0.3\times12=3.6\approx4$ baskets.

Steps:

1. **Split Data into Subsets**: Divide the baskets into smaller subsets that fit into memory.

2. **Apply Apriori Locally**: Run Apriori on each subset to find local frequent itemsets.

3. **Aggregate Results**: Combine the results from all subsets to find global frequent itemsets.

4. **Verification**: Ensure that globally frequent itemsets meet the support threshold across the entire dataset.

By following these steps and applying the SON algorithm, we can handle large datasets efficiently using the MapReduce paradigm.

Step-by-Step SON Algorithm:

Step 1: Divide the Dataset into Subsets

Assume we divide the dataset into 3 subsets, each containing 4 baskets.

Subset 1:

$$\{1,3,4\},\{1,3,5\},\{4,5,6\},\{2,3,6\}$$

Subset 2:

$$\{1,3,5\},\{2,4,6\},\{1,3,4\},\{2,4,5\}$$

Subset 3:

$$\{1,3,6\},\{3,5,6\},\{2,3,6\},\{2,3,5\}$$

Step 2: Run Apriori Algorithm on Each Subset

Subset 1:

- **Frequent Items:** Determine item frequency.

$$\{1:2\},\{3:3\},\{4:2\},\{5:1\},\{6:2\},\{2:1\}$$

  With a local support threshold of 2, the frequent items are: {1, 3, 4, 6}

- **Frequent Pairs:** Generate and count pairs.

$$\{(1,3):2\},\{(3,4):2\},\{(4,6):2\},\{(3,6):2\}$$

Subset 2:

- **Frequent Items:**

$$\{1:2\},\{3:2\},\{5:2\},\{2:2\},\{4:2\},\{6:2\}$$

  Frequent items: {1, 2, 3, 4, 5, 6}

- **Frequent Pairs:**

$$\{(1,3):2\},\{(2,4):2\},\{(4,5):2\}$$

Subset 3:

- **Frequent Items**:

$$\{1:1\},\{3:3\},\{6:3\},\{2:3\},\{5:2\},\{4:1\}$$

  Frequent items: {2, 3, 5, 6}

- **Frequent Pairs**:

$$\{(3,6):3\},\{(2,3):2\},\{(2,6):2\},\{(3,5):2\}$$

Step 3: Aggregate Results from Subsets

Combine the frequent itemsets from all subsets.

Frequent Items Across All Subsets:

$$\{1,2,3,4,5,6\}$$

Frequent Pairs Across All Subsets:

$$\{(1,3):4\},\{(3,4):2\},\{(4,6):2\},\{(3,6):5\},\{(2,4):2\},\{(4,5):2\},\{(2,3):2\},\{(2,$$

Step 4: Determine Global Frequent Itemsets

Using the global support threshold (4), the fre ent pairs are:

Step 4: Determine Global Frequent Itemsets

Using the global support threshold (4), the frequent pairs are:

$$\{(1,3),(3,6)\}$$

Example of MapReduce Steps:

Map Function:

$$\text{Input: } \{\text{Subset 1}\}$$

$$\text{Output: } \{(1,3):2,(3,4):2,(4,6):2,(3,6):2\}$$

Reduce Function:

$$\text{Input: } \{(\text{Key, Value}) \text{ pairs from all mappers}\}$$

$$\text{Output: } \{(1,3):4,(3,6):5\}$$

Parallel Graph Computation

- The Power-Law degree distribution in graphs, where most vertices have few neighbors while a few have many, poses challenges for graph partitioning
-
- High-degree nodes dominate the graph's structure, making it hard to partition without disrupting connectivity.
- High-degree nodes have high centrality, making them crucial for connectivity.
- Power-Law distribution leads to graph fragmentation, making balanced cuts difficult.

1. **High-degree nodes**: Nodes with high degrees have a significant influence on the graph's structure, making it difficult to partition the graph without disrupting the connectivity of these nodes.
2. **Node centrality**: High-degree nodes often have high centrality measures (e.g., betweenness, closeness), making them crucial for the graph's connectivity. Cutting these nodes can lead to a high cost.
3. **Graph fragmentation**: The Power-Law distribution leads to a small number of highly connected nodes, which can cause the graph to fragment into many small components when partitioned. This makes it difficult to find a balanced cut.

As a result, graph partitioning algorithms may perform poorly on Power-Law graphs because they struggle to:

1. **Preserve connectivity**: Algorithms may not be able to preserve the connectivity of high-degree nodes, leading to a high cost.
2. **Balance the cut**: The Power-Law distribution can make it challenging to find a balanced cut, as the majority of nodes have low degrees and are not as influential in the graph's structure.

Q3

Here is a MapReduce pseudocode for the Vertex program implementing a PageRank algorithm:

Mapper

- Emit key-value pairs: (neighbor, 1) for each edge

Reducer

- Initialize PageRank score to 0.15
- Iterate over values (edges)
- Increment PageRank score by the number of edges
- Normalize PageRank score by the number of edges
- Emit updated PageRank score

Note: This is a simplified example and may not be optimized for performance.

Q

Vertex-Cut Partitioning:

1. **Partitioning:** The graph is partitioned into two parts: P1 and P2. The partitioning is done based on the vertex degrees and the graph structure.

P1: {A, B, C, D, E} P2: {F, G, H, I}

2. **Machine 1:** Machine 1 is assigned P1, which contains vertices A, B, C, D, and E.

Machine 1 Execution:

- Machine 1 executes the graph traversal algorithm (e.g., BFS or DFS) on P1.
- The algorithm visits vertices A, B, C, D, and E in the following order: A -> B -> C -> D -> E.
- The algorithm terminates when it reaches the end of the traversal.

Machine 2: Machine 2 is assigned P2, which contains vertices F, G, H, and I.

Machine 2 Execution:

- Machine 2 executes the graph traversal algorithm (e.g., BFS or DFS) on P2.
- The algorithm visits vertices F, G, H, and I in the following order: F -> G -> H -> I.
- The algorithm terminates when it reaches the end of the traversal.

Result: The graph traversal algorithm is executed independently on each machine, and the results are combined to produce the final output.

In this example, the graph is partitioned into two parts, and each part is executed on a separate machine. The graph traversal algorithm is executed independently on each machine, and the results are combined to produce the final output.