20CSGPEBDA:
Big data Analytics.

1BM18CS035
Harsharandhana.k

4) spark RDDs are lazily evaluated, & sometimes we may wish to use RDD multiple times. Spark will recompute RDD & all it dependencies each time we call action RDD. which is expensive for iterative algorithm. which took at data many times. To avoid computing RDD multip times we can use spark to persist data. It will recom-pute lost partitions of data when needed. We can replicate our data on multiple nodes if we want to be able to handle node failure without shutdown.

Level	space used	cpu time	Memory	disk	Comment
MEMORY_ONLY	H	L	Y	Y	
MEMORY_ONLY	L	H	Y	N	
MEMORY_AND_DISK	H	M	Some	Some	split to disk if there is too much data to fit in memory
MEMORY_AND_DISK_SER	L	H	Some	Some	splits to disk if there is too much data to fit memory stores serialized repos
DISK_ONLY	LOW	High	N	Y	

Spark has many layers of persistance to choose from based on what goals are. In python we always serialize the data persist stores. we write data out disk & heap storage, data is always serialized.

1BM18CS035
Harshavardhana.k

part-B

2a)

```
import java.util.Scanner;
object chessking {
    def test(x1: Int, y1: Int, x2: Int, y2: Int):
Boolean =
    {
        if ((x1-x2) >= -1 && (x1-x2) <= 1 &&
            (y1-y2) >= -1 && (y1-y2) <=1)
                return true;
        return false;
    };
def main(args: Array[String]): Unit = {
    var scanner = new Scanner(System.in);
    print("First cell row & column :");
    var x1 = scanner.nextInt();
    var y1 = scanner.nextInt();
    print("Second cell row and column:");

    var x2 = scanner.nextInt();
    var y2 = scanner.nextInt();
    if (test(x1, y1, x2, y2))
        println("Yes");
    else
        println("No");
    }
}
```

1BM18CS035
Harshavardhana.k

26)

```
import scala.collection.mutable.ArrayBuffer;
object fruitsArray {
  def main(args: Array[string]) {

    var fruits = ArrayBuffer[string]
("Apple", "Banana", "Mango", "orange");

fruits += ("strawberry", "pineapple");
scala.util.Sorting.quicksort(Fruits);
fruits = fruits.reverse();

fruits.remove(fruits.length-1, fruit.length-2);
  fruits.toArray;
  }
}
```

1BM18CS035.
Harshavardhana k

2(5) spark-shell

→ val sqlcontext = new org.apache.spark.
→ sql.SQLContext(os)
→ val dfs = sqlContext.read.json("cars.json")

→ dfs.select("carName", "carprice").orderBy("carprice")
.show(10)

→ dfs.select(dfs.columns(10)).show().

1BM18CS035.
Harshavardhan.k

3b)

Tuple: is basically grouping data in of different types:

* To create tuple we can do val tup = (42, "Jam")
* To access member of tuple tup._1
* To copy & change some values:
  tup.copy(_1 = 0)
* To pretty print them tup.toString.
* To swap values tup.swap.

Maps: are useful data structure to make associations between keys and values.

* We can create map like val map = Map("Has" → 234, "Nas" → 78)
* Check values exist in map or map.contain("Bob")
* Access values in map. map("Has")
* Add new pairing to map by creating new one val anothermap = map + ("Bob", 45)

⇒ program:

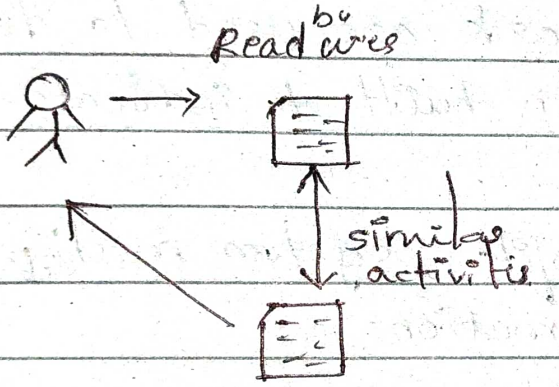
```
def getCapital(country: string, cmap:
  Map[string, string]; string = {
    if (cmap.contains(country)){
      return.cmap.get(country);
    }
  return cmap.get("India");
}
```

IBM18CS035
Harshavardhana.k

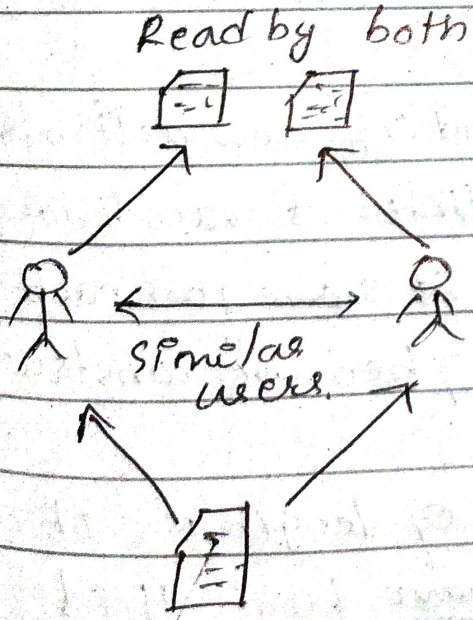
part-c.

9a) Two types are:

* content based filtering: focus on properties of items, similarity of items is determined by measuring similarities in their properties.
* collaborative based filtering: focus on relationship b/w users & items. similarity of items is determin by similarity of rating of those items by users who have rated both items.



← content Based filtering.



← collaborative filtering.

Content based Recommendations / Filtering:

* Based on description of item & profile of user's preference.
* This method are best suited to situations where there is known data on item but not on user.
* They treat recommendation as user-specific classification problem & learn from classifiers for user's like & dislike based on item's

In this system keywords are used to describe items, & user profile is built to indicate type of item this used likes.

To create a user profile, system mostly focus on two types of information:

* Model of user's preferences
* history of user's interaction with recommender system.

Consider an example of recommending new action to user. Lets say we have 100 articles & vocabulary of size N. We first compare tf-id score for each of words for every article. Then we construct 2 vectors:

1) Item vector: this is a vector of length N. It contains 1 for words that have high tf-id score in that article otherwise 0.

	bacon	beans	beautiful	blue	breakfast	brown	dog
0	0.00	0.00	0.60	0.53	0.00	0.00	0.00
1	0.00	0.00	0.49	0.43	0.00	0.00	0.00

2) User vector: Again 1xN vector, we store probability of word occuring, in articles that user has consumed once we have these profiles, we compute similarity between users and items. 1) User has highest similarity 2) has highest similarity with other items that user has read.

2 common methods: we simply take cosine similarity between user & item vector to obtain user-item similarity. To recommend items that are most similar to items that user brought, we compute the cosine similarity.

$$\text{cosine}(x,y) = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$$

Jaccad similarity: intersection over union, formula

$$J(X,Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

this is used for item similarity. we compare item vectors with each other & return items that are most similar.

Jaccad similarity is useful only when the vectors contain binary values. If they have ranking of rating that can take on multiple values. Jaccad similarity is not applicable.