

```

AI 1
t = {
    1: [2, 3],
    2: [4, 5],
    3: [6],
    4: [],
    5: [],
    6: []
}

def dfs(node):
    print(node, end=' ')
    for child in t[node]:
        dfs(child)

print("DFS Traversal: ")
dfs(1)

```

```

AI 2
from collections import deque

t = {
    1: [2, 3],
    2: [4, 5],
    3: [6],
    4: [],
    5: [],
    6: []
}

def bfs(root):
    queue = deque([root])
    while queue:
        node = queue.popleft()
        print(node, end=' ')
        for child in t[node]:
            queue.append(child)

print("BFS Traversal: ")
bfs(1)

```

```

AI 3
def water_jug():
    x, y = 0, 0
    X, Y, Z = 4, 3, 2

    print(f"Initial State: ({x},{y})")

    while True:
        if x == Z or y == Z:
            print(f"Goal Reached: ({x},{y})")
            break

        elif x == 0:
            x = X
            print(f"Fill Jug 1: ({x},{y})")

        elif y != Y:
            pour = min(x, Y - y)
            x = x - pour
            y = y + pour
            print(f"Pour Jug 1 -> Jug 2: ({x},{y})")

        elif y == Y:

```

```

y = 0
print(f"Empty Jug 2: ({x},{y})")

```

water\_jug()

AI 5

```

from itertools import permutations

```

```

d = [
    [0, 10, 15, 20],
    [10, 0, 35, 25],
    [15, 35, 0, 30],
    [20, 25, 30, 0]
]

```

```

cities = [0, 1, 2, 3]

```

```

min_cost = 999999

```

```

best_path = None

```

```

for p in permutations(cities):
    cost = 0
    for i in range(len(p) - 1):
        cost += d[p[i]][p[i + 1]]
    cost += d[p[-1]][p[0]] # return to starting city

    if cost < min_cost:
        min_cost = cost
        best_path = p

```

```

print("Best Path:", best_path)
print("Minimum Cost:", min_cost)

```

AI 6

```

def hanoi(n, source, helper, destination):
    if n == 1:
        print(f"Move disk 1 from {source} to {destination}")
        return

    hanoi(n-1, source, destination, helper)
    print(f"Move disk {n} from {source} to {destination}")
    hanoi(n-1, helper, source, destination)

```

```

n = 3

```

```

hanoi(n, "A", "B", "C")

```

BD.21

Practical No: 21

Steps:

1. Download and install JDK (Java 8 or 11).
2. Verify installation: java -version using cmd.
3. Download Hadoop binary (e.g., Hadoop 3.x) from Apache Hadoop website & extract it in: C:\hadoop.
4. Add in System Environment Variables:  
HADOOP\_HOME= C:\hadoop  
Add to PATH:  
C:\hadoop\bin  
C:\hadoop\sbin
5. Verify Hadoop is installed or not: hadoop version
6. Configure Hadoop Files  
Go to:  
C:\hadoop\etc\hadoop

```

1) core-site.xml
Open and add:
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
</configuration>

2) hdfs-site.xml
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>
7. Format NameNode in cmd
hdfs namenode -format
8. Start Hadoop Services: start-dfs.cmd
Check in browser:
http://localhost:9870
9. Create Directory in HDFS: hdfs dfs -mkdir /mydata
   hdfs dfs -ls /
10. Create data.txt in C drive with content:
Big Data Hadoop Practical
11. Upload File to HDFS: hdfs dfs -put C:\data.txt /mydata
Verify: hdfs dfs -ls /mydata
12. View File in HDFS: hdfs dfs -cat /mydata/data.txt
13. Download File from HDFS to Local:
hdfs dfs -get /mydata/data.txt C:\hdfs_output.txt
Check in C drive for hdfs_output.txt
14. Rename File in HDFS
hdfs dfs -mv /mydata/data.txt /mydata/bigdata.txt
Check:
hdfs dfs -ls /mydata
Step 13: Delete File from HDFS
hdfs dfs -rm /mydata/bigdata.txt
Verify:
hdfs dfs -ls /mydata

```

BD.22

Aim: Write a MapReduce program to count word frequencies from a large dataset stored in HDFS. Execute and analyze the job logs to identify the mapper and reducer outputs.

Code:

WordCount.java

```

import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

    public static class MapperClass
        extends Mapper<Object, Text, Text, IntWritable> {

        public void map(Object key, Text value, Context context)
            throws IOException, InterruptedException {

```

```

        StringTokenizer st = new StringTokenizer(value.toString());
        while (st.hasMoreTokens()) {
            context.write(new Text(st.nextToken()), new IntWritable(1));
        }
    }
}
public static class ReducerClass
    extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values,
        Context context) throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values)
            sum += val.get();
        context.write(key, new IntWritable(sum));
    }
}
public static void main(String[] args) throws Exception {

    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Word Count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(MapperClass.class);
    job.setReducerClass(ReducerClass.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}

```

BD.23

```

1. use CollegeDB
2. db.createCollection("students")
3. db.students.insertMany([
    {roll: 1, name: "Amit", age: 20, dept: "CS", marks: 85},
    {roll: 2, name: "Neha", age: 21, dept: "IT", marks: 78},
    {roll: 3, name: "Rahul", age: 22, dept: "CS", marks: 90},
    {roll: 4, name: "Pooja", age: 20, dept: "EXTC", marks: 88},
    {roll: 5, name: "Rohan", age: 23, dept: "CS", marks: 72},
    {roll: 6, name: "Ananya", age: 21, dept: "IT", marks: 95}
])
4. db.students.find().pretty()
5. db.students.find({}, {name:1, dept:1, _id:0})
6. db.students.find({name: "Rahul"}).pretty()
7. db.students.find({marks: {$gt: 80}}).pretty()
8. db.students.find({dept: "CS"}).pretty()
9. db.students.find({age: {$gte: 21}}).pretty()
10. db.students.aggregate([
    {$group: {_id: null, avgMarks: {$avg: "$marks"}}}
])
11. db.students.aggregate([
    {$group: {_id: "$dept", total: {$sum: 1}}}
])
12. db.students.aggregate([
    {$group: {_id: null, maxMarks: {$max: "$marks"}}}
])

```