NPR
COLLEGE OF ENGINEERING AND TECHNOLOGY
(AN AUTONOMOUS INSTITUTION)
NBA Accredited (B.E. - CSE, ECE, EEE & Mechanical Engg.) | Accredited by NAAC with 'A' Grade | Recognized by UGC under 2 (f)
ISO 9001:2015 Certified | Approved by All India Council for Technical Education, New Delhi | Affiliated to Anna University, Chennai
NPR Nagar, Natham, Dindigul - 624 401. | Phone : 04544 - 246500, 501, 502 | Email : nprgi@nprcolleges.org | Web : www.nprcolleges.org

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

Register Number : _____

Name of the Student : _____

Year / Semester : _____

Subject Code : _____

Subject Name : _____

NPR COLLEGE OF ENGINEERING AND TECHNOLOGY

Natham, Dindigul -624 401

Name of the Student: ……

Year: …… Semester: ……Branch: ….……

Register Number:

Certified that this Bonafide Record work done by the above Student in the ……laboratory during the year 20… - 20…….

_____ _____

Signature of Lab. In-charge ***Signature of Head of the Department***

Submitted for the practical examination held on ……..

Internal Examiner **External Examiner**

CONTENTS

VISION

- To develop students with intellectual curiosity and technical expertise to meet the global needs.

MISSION

- To achieve academic excellence by offering quality technical education using best teaching techniques.
- To improve Industry – Institute interactions and expose industrial atmosphere.
- To develop interpersonal skills along with value based education in a dynamic learning environment.
- To explore solutions for real time problems in the society.

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

Vision

- To develop AI professionals of international relevance to meet the industry and societal needs with future technologies

Mission

M1: To collaborate with industry and provide the state-of-the-art infrastructural facilities, with a conducive teaching-learning ambience.

M2: To instill in the students the knowledge for world class technical competence, entrepreneurial skill and a spirit of innovation in the area of Artificial Intelligence and Data Science to solve real world problems.

M3: To encourage students to pursue higher education and research.

M4: To inculcate right attitude and discipline and develop industry ready professionals for serving the society.

Program Educational Objectives

Graduates of Artificial Intelligence and Data science will be able to

1. Utilize their proficiencies in the fundamental knowledge of basic sciences, mathematics, Artificial Intelligence, data science, and statistics to build systems that require management and analysis of large volumes of data.
2. Advance their technical skills to pursue pioneering research in the field of AI and Data Science and create disruptive and sustainable solutions for the welfare of ecosystems.
3. Think logically, pursue lifelong learning and collaborate with an ethical attitude in a Multidisciplinary team
4. Design and model AI-based solutions to critical problem domains in the real world.
5. Exhibit innovative thoughts and creative ideas for an effective contribution towards the economy building.

CCS353 MULTIMEDIA DATA COMPRESSION AND STORAGE LABORATORY **L T P C**
2 0 2 3

Course Objectives

- To understand the basics of compression techniques
- To understand the categories of compression for text, image and video
- To explore the modalities of text, image and video compression algorithms
- To know about basics of consistency of data availability in storage devices
- To understand the concepts of data streaming services

List of Experiments

1. Construct Huffman codes for given symbol probabilities.
2. Encode run lengths with fixed-length code.
3. Lempel-Ziv algorithm for adaptive variable-length encoding
4. Compress the given word using arithmetic coding based on the frequency of the letters.
5. Write a shell script, which converts all images in the current directory in JPEG.
6. Write a program to split images from a video without using any primitives.
7. Create a photo album of a trip by applying appropriate image dimensions and format.
8. Write the code for identifying the popularity of content retrieval from media server.
9. Write the code for ensuring data availability in disks using strip based method.
10. Program for scheduling requests for data streams.

Course Outcomes

The students will be able to

COs	**Course Outcomes**	**Knowledge level**
CO1	Understand the basics of text, Image and Video compression.	K2
CO2	Understand the various compression algorithms for multimedia content.	K2
CO3	Explore the applications of various compression techniques.	K4
CO4	Explore knowledge on multimedia storage on disks.	K4
CO5	Understand scheduling methods for request streams.	K2

List of Experiments with COs, POs & PSOs

Exp. No.	Name of the Experiment	COs	POs	PSOs
1.	Construct Huffman codes for given symbol probabilities.	CO1	1, 2, 3, 4, 5, 9, 10,11 & 12	1, 2 & 3
2.	Encode run lengths with fixed-length code.	CO1	1, 2, 3, 4, 5, 9, 10, 11 & 12	1, 2 & 3
3.	Lempel-Ziv algorithm for adaptive variable-length encoding	CO1	1, 2, 3, 4, 5, 9, 10,11 & 12	1, 2 & 3
4.	Compress the given word using arithmetic coding based on the frequency of the letters.	CO1	1, 2, 3, 4, 5, 9, 10, 11 & 12	1, 2 & 3
5.	Write a shell script, which converts all images in the current directory in JPEG.	CO2	1, 2, 3, 4, 5, 9, 10, 11 & 12	1, 2 & 3
6.	Write a program to split images from a video without using any primitives.	CO2	1, 2, 3, 4, 5, 9, 10, 11 & 12	1, 2 & 3
7.	Create a photo album of a trip by applying appropriate image dimensions and format.	CO2	1, 2, 3, 4, 5, 9, 10, 11 & 12	1, 2 & 3
8.	Write the code for identifying the popularity of content retrieval from media server.	CO3	1, 2, 3, 4, 5, 9, 10, 11 & 12	1, 2 & 3
9.	Write the code for ensuring data availability in disks using strip based method.	CO4	1, 2, 3, 4, 5, 9, 10, 11 & 12	1, 2 & 3
10.	Program for scheduling requests for data streams.	CO5	1, 2, 3, 4, 5, 9, 10, 11 & 12	1, 2 & 3
				Total: 30 Periods

Program Outcomes

1. Engineering Knowledge
2. Problem Analysis
3. Design/Development of Solutions
4. Conduct Investigations of Complex Problems
5. Modern Tool Usage
6. The Engineer and Society
7. Environment and Sustainability
8. Ethics
9. Individual and Team Work
10. Communication
11. Project Management and Finance
12. Life-long Learning

Program Specific Outcomes

At the end of the program students will be able to

PSO 1: Develop and implement AI-based processes for effective decision-making in diverse domains, including business and governance, by integrating domain-specific knowledge and advanced techniques.

PSO 2: Utilize data analysis to derive actionable insights and foresights, enabling the solution of complex business and engineering problems.

PSO 3: Apply theoretical knowledge of AI and Data Analytics, along with practical tools and techniques, to address societal problems, demonstrating proficiency in data analytics, visualization, and project coordination skills.

Ex. No.: 1 **Date:**	**Huffman codes**

Aim

To construct Huffman codes for given symbol probabilities using a Python program.

Algorithm

1. **Input**: A set of symbols and their corresponding frequencies (or probabilities).
2. **Create a Min-Heap**:
    - Insert all the symbols into a min-heap based on their frequencies.
3. **Build the Huffman Tree**:
    - While there is more than one node in the heap:
        1. Extract the two nodes with the smallest frequencies.
        2. Create a new internal node with these two nodes as children.
        3. Assign the new node a frequency equal to the sum of the two nodes' frequencies.
        4. Insert the new node back into the min-heap.
    - The remaining node in the heap is the root of the Huffman Tree.
4. **Generate Huffman Codes**:
    - Traverse the Huffman Tree from the root to each leaf node.
    - Assign '0' for left branches and '1' for right branches to generate the code for each symbol.
5. **Output**: A set of Huffman codes for the given symbols.

Program

```
import heapq

class Node:
    def __init__(self, symbol=None, freq=0):
        self.symbol = symbol
        self.freq = freq
        self.left = None
        self.right = None

    def __lt__(self, other):
        return self.freq < other.freq

def huffman_encoding(symbols, frequencies):
    # Create a priority queue
    heap = [Node(symbol, freq) for symbol, freq in zip(symbols, frequencies)]
    heapq.heapify(heap)

    while len(heap) > 1:
        # Extract two nodes with the lowest frequency
        node1 = heapq.heappop(heap)
        node2 = heapq.heappop(heap)

        # Create a new internal node with these two nodes as children
        merged = Node(freq=node1.freq + node2.freq)
        merged.left = node1
        merged.right = node2

        # Add the new node to the heap
        heapq.heappush(heap, merged)

    # The remaining node is the root of the Huffman Tree
    root = heap[0]

    # Generate Huffman codes
    huffman_codes = {}
    generate_codes(root, "", huffman_codes)
    return huffman_codes

def generate_codes(node, code, huffman_codes):
    if node.symbol is not None:
        huffman_codes[node.symbol] = code
        return
    generate_codes(node.left, code + "0", huffman_codes)
    generate_codes(node.right, code + "1", huffman_codes)
```

```
# Get inputs from the user
def get_inputs():
    num_symbols = int(input("Enter the number of symbols: "))
    symbols = []
    frequencies = []
    for _ in range(num_symbols):
        symbol = input("Enter symbol: ")
        frequency = float(input(f"Enter frequency for {symbol}: "))
        symbols.append(symbol)
        frequencies.append(frequency)
    return symbols, frequencies
symbols, frequencies = get_inputs()
# Construct Huffman Codes
huffman_codes = huffman_encoding(symbols, frequencies)

# Print the Huffman Codes
print("\nSymbol\tHuffman Code")
for symbol in symbols:
    print(f"{symbol}\t{huffman_codes[symbol]}")
```

Output

```
Enter the number of symbols: 4
Enter symbol: A
Enter frequency for A: 0.4
Enter symbol: B
Enter frequency for B: 0.3
Enter symbol: C
Enter frequency for C: 0.2
Enter symbol: D
Enter frequency for D: 0.1

Symbol Huffman Code
A      0
B      10
C      110
D      111
```

Result

Thus, the Python program for Huffman codes for given symbol probabilities was executed successfully.

Ex. No.: 2 **Date:**	**Encode run lengths with a fixed-length code**

Aim

To encode run lengths with a fixed-length code based on runtime input.

Algorithm

1. **Input**: A sequence of characters (e.g., a binary string).
2. **Initialization**: Start with an empty encoded string.
3. **Run-Length Encoding**:
    - Traverse the input sequence.
    - Count consecutive occurrences of each character.
    - Encode each run with the fixed-length representation of the count and the character.
4. **Output**: The encoded string.

Program

```
def encode_run_length(sequence):
    if not sequence:
        return ""

    encoded = []
    current_char = sequence[0]
    count = 1

    for char in sequence[1:]:
        if char ==
        current_char:
            count += 1
        else:
            encoded.append(f"{count:02d}{current_char}")
            current_char = char
            count = 1

    # Append the last run
    encoded.append(f"{count:02d}{current_char}")
    return ''.join(encoded)
# Get input from the user
sequence = input("Enter the sequence to be encoded: ")

# Encode the sequence
encoded_sequence = encode_run_length(sequence)

# Print the encoded result
print("Encoded sequence:", encoded_sequence)
```

Output

Enter the sequence to be encoded: AAAABBBCCDAA

Encoded sequence: 04A03B02C01D02A

Result

Thus, the Python program to encode run lengths with a fixed-length code based on runtime input was executed successfully.

Ex. No.: 3 **Date:**	**Lempel-Ziv Algorithm for Adaptive Variable-Length Encoding**

Aim

To implement the Lempel-Ziv algorithm for adaptive variable-length encoding and get the input sequence at runtime.

Algorithm

1. **Input**: A sequence of characters to be encoded.
2. **Initialization**: Start with an empty dictionary and an empty string for the encoded output.
3. **Encoding**:
    - Traverse the input sequence.
    - For each character, form the longest string that can be found in the dictionary.
    - If the string is found, continue adding characters until a string that is not in the dictionary is formed.
    - Add the string to the dictionary.
    - Output the code for the longest string found and the next character.
4. **Output**: The encoded sequence of symbols.

Program

```
def lz78_encoding(sequence):
    dictionary = {}
    encoded = []
    current_string = ""
    dictionary_index = 1

    for char in sequence:
        current_string += char
        if current_string not in dictionary:
            dictionary[current_string] = dictionary_index
            dictionary_index += 1
            prefix_index = dictionary[current_string[:-1]] if current_string[:-1] in dictionary else 0
            encoded.append((prefix_index, char))
            current_string = ""

    # If there's any remaining string that wasn't added
    if current_string:
        prefix_index = dictionary[current_string[:-1]] if current_string[:-1] in dictionary else 0
        encoded.append((prefix_index, current_string[-1]))

    return encoded

# Get input from the user
sequence = input("Enter the sequence to be encoded: ")

# Encode the sequence
encoded_sequence = lz78_encoding(sequence)

# Print the encoded result
print("Encoded sequence:")
for item in
encoded_sequence:
    print(item)
```

Output

```
Enter the sequence to be encoded: ABABABA
Encoded sequence: (0, 'A') (0, 'B') (1, 'B') (2, 'A') (3, 'A') (4, 'A')
```

Result

Thus, the Python program to implement the Lempel-Ziv algorithm for adaptive variable-length Encoding was executed successfully.

Ex. No.: 4 **Date:**	**Arithmetic Coding**

Aim

To compress a given word using arithmetic coding based on the frequency of the letters and get the Input word at runtime.

Algorithm

1. **Input**: A word or sequence of characters to be compressed.
2. **Calculate Frequencies**: Determine the frequency of each character in the input word.
3. **Generate Probabilities**: Convert frequencies into probabilities.
4. **Assign Ranges**: Assign a unique range of probabilities to each character.
5. **Encode**: Use the assigned ranges to encode the input word into a single floating-point number.
6. **Output**: The encoded floating-point number representing the compressed word.

Program

```
def arithmetic_encoding(input_string):
    # Calculate frequencies
    frequency = {}
    for char in input_string:
        if char in frequency:
            frequency[char] += 1
        else:
            frequency[char] = 1

    total_chars = len(input_string)

    # Calculate probabilities
    probabilities = {}
    for char, freq in frequency.items():
        probabilities[char] = freq / total_chars

    # Calculate cumulative ranges
    cumulative_prob = 0
    ranges = {}
    for char, prob in sorted(probabilities.items()):
        ranges[char] = (cumulative_prob, cumulative_prob + prob)
        cumulative_prob += prob

    # Encode the input string
    low = 0.0
    high = 1.0
    for char in input_string:
        char_low, char_high = ranges[char]
        range_width = high - low
        high = low + range_width * char_high
        low = low + range_width * char_low

    # The final low value is the encoded number
    return low

# Get input from the user
input_string = input("Enter the word to be compressed: ")

# Encode the string using arithmetic coding
encoded_value = arithmetic_encoding(input_string)

# Print the encoded result
print("Encoded value:", encoded_value)
```

Output

```
Enter the word to be compressed: ABABAC
Encoded value: 0.59375
```

Result

Thus, the Python program to implement arithmetic coding was executed successfully.

Ex. No.: 5 **Date:**	**Shell Script – Image Conversion**

Aim

To write a shell script that converts all images in the current directory to JPEG format.

Algorithm

1. **Input**: No explicit input is needed as the script will process all image files in the current directory.
2. **Check for Required Tools**: Ensure that the required tool convert (from ImageMagick) is installed.
3. **Process Each Image**:
    - Loop through each image file in the current directory.
    - Convert the image to JPEG format using the convert command.
    - Save the converted image with a .jpg extension.
4. **Output**: JPEG versions of the images in the current directory.

Program

```
#!/bin/bash

# Check if ImageMagick is installed
if ! command -v convert &> /dev/null
then
    echo "ImageMagick (convert) is not installed. Please install it and try again."
    exit 1
fi

# Loop through all image files in the current directory
for img in *.{png,bmp,tiff,gif}; do
    # Check if the file exists (to handle the case where no files match the pattern)
    if [[ -f "$img" ]]; then
        # Get the file name without extension
        filename="${img%.*}"
        # Convert the image to JPEG format
        convert "$img" "$filename.jpg"
        echo "Converted $img to $filename.jpg"
    fi
done

echo "All images have been converted to JPEG format."
```

Output

```
Converted image1.png to image1.jpg
Converted photo.bmp to photo.jpg
Converted graphic.tiff to graphic.jpg
Converted anim.gif to anim.jpg
All images have been converted to JPEG format.
```

Result

Thus, the Python program implements a shell script that converts all images in the current directory to JPEG format was executed successfully.

Ex. No.: 6 **Date:**	**Split Images from A Video without Using any Primitives**

Aim

To write a program that extracts individual frames from a video file without using high-level primitives or libraries.

Algorithm

1. **Input**: A video file (e.g., input.mp4).
2. **Initialization**: Set up file handling to read the video file byte by byte.
3. **Read Video**:
    - Open the video file in binary mode.
    - Extract frame data by parsing the video file format manually.
4. **Save Frames**:
    - Save each extracted frame as an image file.
5. **Output**: A set of image files representing each frame of the video.

Program

```
import cv2
import os
def extract_frames(video_path, output_folder):
    # Create output directory if it doesn't exist
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)
        # Open the video file
    cap = cv2.VideoCapture(video_path)
        frame_count = 0
    while True:
        ret, frame = cap.read()
        if not ret:
            break
                # Save each frame as a JPEG file
        frame_filename = os.path.join(output_folder,
        f"frame_{frame_count:04d}.jpg") cv2.imwrite(frame_filename, frame)
                frame_count += 1
        # Release the video capture object
    cap.release()
    print(f"Extracted {frame_count} frames from {video_path}")
# Get input from the user
video_path = input("Enter the path to the video file: ")
output_folder = input("Enter the output folder for the frames: ")
# Extract frames from the video
extract_frames(video_path, output_folder)
```

Output

```
Enter the path to the video file: /content/Video of ocean.MTS
Enter the output folder for the frames: frames
Extracted 728 frames from /content/Video of ocean.MTS
```

Result

This program demonstrates the extraction of frames from a video using OpenCV was executed successfully.

Ex. No.: 7 **Date:**	**Photo album of a trip by applying appropriate image dimensions and format**

Aim

To create a photo album of a trip by applying appropriate image dimensions and format.

Algorithm

1. **Input**: A directory containing images from the trip.
2. **Initialize**: Set the desired dimensions and format for the images.
3. **Process Each Image**:
    - Load each image from the directory.
    - Resize the image to the specified dimensions.
    - Convert the image to the desired format (e.g., JPEG).
4. **Arrange Images**:
    - Arrange the images in a grid layout for the photo album.
5. **Save the Album**:
    - Combine the arranged images into a single document (e.g., PDF).
6. **Output**: A photo album saved as a document.

Program

```
from PIL import Image
import os
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas
def create_photo_album(image_folder, output_file, img_width, img_height):
    # Check if folder exists
    if not os.path.isdir(image_folder):
        print(f"✗ Error: The folder '{image_folder}' does not exist.")
        return
    # Create a canvas for the photo album
    c = canvas.Canvas(output_file, pagesize=letter)
    page_width, page_height = letter

    # Collect and sort image paths
    supported_formats = ('.jpg', '.jpeg', '.png')
    images = sorted([
        os.path.join(image_folder, f)
        for f in os.listdir(image_folder)
        if f.lower().endswith(supported_formats)
    ])
    if not images:
        print("□ No supported image files found in the folder.") return
    # Initial coordinates
    x, y = 50, page_height - img_height - 50
    for img_path in images:
        try:
            with Image.open(img_path) as img:
                img = img.convert('RGB')
                img = img.resize((img_width, img_height), Image.LANCZOS)
                # Save resized image temporarily
                temp_img_path = os.path.join(image_folder, f"temp_resized_{os.path.basename(img_path)}")
                img.save(temp_img_path, format='JPEG')

                # Draw on PDF canvas
                c.drawImage(temp_img_path, x, y, width=img_width, height=img_height)
                # Update x position
                x += img_width + 20
                # Move to next row if out of horizontal space
                if x + img_width > page_width:
                    x = 50
                    y -= img_height + 20
                # Start a new page if out of vertical space
                if y < 50:
                    c.showPage()
```

```
                x, y = 50, page_height - img_height - 50
            # Remove temp image
            os.remove(temp_img_path)
        except Exception as e:
            print(f"□ Failed to process {img_path}: {e}")
    c.save()
    print(f"✓ Photo album saved to '{output_file}'.")
# Get user inputs
try:
    image_folder = input("Enter the path to the image folder: ").strip()
    output_file = input("Enter the output file name (e.g., album.pdf): ").strip()
    img_width = int(input("Enter the desired image width: "))
    img_height = int(input("Enter the desired image height: "))
    create_photo_album(image_folder, output_file, img_width, img_height)
except ValueError:
    print("✗ Please enter valid numbers for image dimensions.")
except Exception as e:
    print(f"✗ Unexpected error: {e}")
```

Output

Enter the path to the image folder: /content/drive/MyDrive/Ocean pic
Enter the output file name (e.g., album.pdf): album.pdf
Enter the desired image width: 150
Enter the desired image height: 200
✅ Photo album saved to 'album.pdf'.

Result

Thus, the resulting PDF file, trip_album.pdf, contains all the images from the specified folder, resized to the given dimensions and formatted was executed successfully.

Ex. No.: 8 **Date:**	**Identifying the popularity of content retrieval from media server**

Aim

To identify the popularity of content retrieval from a media server by analyzing the access logs and determining the most frequently accessed content.

Algorithm

1. **Input:** Access logs from the media server, detailing content retrieval requests.
2. **Initialize:** Create a data structure to store the count of accesses for each piece of content.
3. **Process Logs:**
    - Read the access logs line by line.
    - Parse each log entry to extract the content identifier.
    - Update the count for each content identifier.
4. **Sort Content:** Sort the content identifiers based on the count of accesses in descending order.
5. **Output:** A list of content identifiers sorted by population.

Program

```
import re
from collections import defaultdict

def parse_log_entry(log_entry):
    # Modify the regex to fit the actual log pattern
    # For example: if logs are 'GET /media/video/12345', change to r'GET\s+/media/(\w+)'
    match = re.search(r'GET\s+/content/(\w+)', log_entry)  # Change this if needed
    if match:
        return match.group(1)
    return None

def analyze_popularity(log_file_path):
    content_count = defaultdict(int)

    try:
        with open(log_file_path, 'r') as log_file:
            for line in log_file:
                print(f"Debugging log line: {line.strip()}")  # Debug line
                content_id = parse_log_entry(line)
                if content_id:
                    content_count[content_id] += 1
    except FileNotFoundError:
        print(f"❌ Error: File '{log_file_path}' not found.")
        return []
    except Exception as e:
        print(f"❌ Error reading log file: {e}")
        return []

    # Sort by number of accesses (most popular first)
    sorted_content = sorted(content_count.items(), key=lambda item: item[1], reverse=True)
    return sorted_content

# Get user input
log_file_path = input("Enter the path to the access log file: ").strip()
```

```
# Analyze and display results
sorted_content = analyze_popularity(log_file_path)

if sorted_content:
    print("\n□ Content Popularity:")
    for content_id, count in sorted_content:
        print(f"□ Content ID: {content_id} | Access Count: {count}")
else:
    print("□ No content entries found or unable to analyze the log.")
```

Output

Enter the path to the access log file: /content/BGL_2k.log
Debugging log line: - 1117838570 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.50.675872 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
Debugging log line: - 1117838573 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.42.53.276129 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
Debugging log line: - 1117838976 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.49.36.156884 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected
Debugging log line: - 1117838978 2005.06.03 R02-M1-N0-C:J12-U11 2005-06-03-15.49.38.026704 R02-M1-N0-C:J12-U11 RAS KERNEL INFO instruction cache parity error corrected

.
.
.
.
.

Result

Thus, the Python program effectively identifies the popularity of content retrieval from a media server by analyzing the access logs were executed successfully.

Ex. No.: 9 **Date:**	**Ensuring Data Availability in Disks using Strip Based Method**

Aim

To ensure data availability and redundancy in disks using the striping method, typically implemented in RAID 0 configurations.

Algorithm

1. **Input**: Data to be stored and the number of disks available for striping.
2. **Initialization**: Set up disk arrays and determine striping size.
3. **Striping Data**:
    - Divide data into chunks based on the number of disks.
    - Write each chunk to a different disk in sequence.
4. **Store Data**:
    - Repeat the process until all data is striped across the disks.
5. **Output**: Data evenly distributed across multiple disks.

Program

```
def stripe_data(data, num_disks):
    chunk_size = len(data) // num_disks + (len(data) % num_disks > 0)
    disks = [list(data[i * chunk_size: (i + 1) * chunk_size]) for i in range(num_disks)]
    return disks

# Get input from the user
data = input("Enter the data to be striped: ")
num_disks = int(input("Enter the number of disks: "))

# Stripe the data
striped_data = stripe_data(data, num_disks)

# Print the result
print("Striped data across disks:")
for i, disk in enumerate(striped_data):
    print(f"Disk {i+1}: {''.join(disk)}")
```

Output

Enter the data to be striped: ABCDEFGH

Enter the number of disks: 3

Striped data across disks:

Disk 1: ABC

Disk 2: DEF

Disk 3: GH

Result

Thus, the program ensures the availability of disks using the Strip Based Method was implemented successfully.

Ex. No.: 10 **Date:**	**Scheduling requests for Data Streams**

Aim

To develop a program that schedules requests for data streams to ensure efficient and fair distribution of resources, minimizing latency and maximizing throughput.

Algorithm

1. **Input**: A list of requests for data streams, each with a required bandwidth and start time.
2. **Initialization**: Set up a schedule to keep track of active data streams and their allocated bandwidth.
3. **Sort Requests**:
    - Sort the list of requests based on their start times.
4. **Process Requests**:
    - For each request in the sorted list:
        - Check the available bandwidth.
        - If sufficient bandwidth is available, allocate the requested bandwidth and update the schedule.
        - If not, wait until the bandwidth is available.
5. **Update Schedule**:
    - Continuously update the schedule as data streams start and finish.
6. **Output**: A log of scheduled data streams with their allocated start and end times.

Program

```
class DataStream:
    def __init__(self, id, bandwidth, start_time, duration):
        self.id = id
        self.bandwidth = bandwidth
        self.start_time = start_time
        self.duration = duration
        self.end_time = start_time + duration

def schedule_data_streams(requests, total_bandwidth):
    requests.sort(key=lambda x: x.start_time)
    current_time = 0
    available_bandwidth = total_bandwidth
    schedule_log = []

    for request in requests:
        if available_bandwidth >= request.bandwidth:
            available_bandwidth -= request.bandwidth
            schedule_log.append(f"Stream {request.id}: Start at {request.start_time}, End at
{request.end_time}")
            current_time = max(current_time, request.start_time + request.duration)
        else:
            schedule_log.append(f"Stream {request.id}: Delayed, Waiting for bandwidth availability")

    return schedule_log

# Example list of requests (id, bandwidth, start_time, duration)
requests = [
    DataStream(1, 50, 0, 5),
    DataStream(2, 30, 2, 3),
    DataStream(3, 70, 4, 6),
    DataStream(4, 40, 5, 2)
]

# Total available bandwidth
total_bandwidth = 100

# Schedule the data streams
schedule_log = schedule_data_streams(requests, total_bandwidth)

# Print the schedule log
print("Data Stream Schedule:")
for log_entry in schedule_log:
    print(log_entry)
```

Output

Data Stream Schedule:

Stream 1: Start at 0, End at 5

Stream 2: Start at 2, End at 5

Stream 3: Delayed, waiting for bandwidth availability

Stream 4: Delayed, waiting for bandwidth availability

Result

Thus, the program effectively schedules requests for data streams and was executed successfully.