ಭಾರತೀಯ ಮಾಹಿತಿ ತಂತ್ರಜ್ಞಾನ ಸಂಸ್ಥೆ ರಾಯಚೂರು
भारतीय सूचना प्रौद्योगिकी संस्थान रायचूर
Indian Institute of Information Technology Raichur

Lab - 5
Introduction to Programming (ID110)
Date: November 23, 2024
Topics: Pointers, 1-D & 2-D Arrays

Time: 1.5 Hr | CSE'24, Semester - I | Max marks: 10

Instructions:

- The lab session consists of **two programming questions**, and both of them are **mandatory**.
- External materials (e.g., notes, books) and electronic devices (e.g., mobile phones, smartwatch, Bluetooth) are **strictly prohibited**. Only a **blank sheet of paper** and a **pen** may be used for rough work.
- Internet usage is **not allowed** under any circumstances. Any violations will lead to **serious academic consequences**, including potential disqualification from the lab.
- Any form of **plagiarism or academic dishonesty** will be treated with the utmost seriousness and may result in severe penalties, including a zero for the lab or further disciplinary actions.
- Code must be written from scratch during the session. Pre-written code snippets or solutions will not be accepted. Use meaningful variable names and add appropriate comments where necessary.
- Upon completion, **two code files** named after **roll no.** (e.g., `"CS24B1001-Lab5-p1.c"` and `"CS24B1001-Lab5-p2.c"`) must be submitted on **Google Classroom**. Not following the naming convention will lead to **minus marking**. The submission will only be accepted if done in the presence of TA.

---

1. You are given a **grayscale image matrix** represented as a 2D array where each element of the matrix is an integer between 0 and 255, representing the intensity of the pixel. Write a program that takes the image matrix and an **image filter (given as an integer code)** as input and outputs the resulting matrix after applying the filter.

   The program should support the following filters:

   1. **Box Blur (Code: 1)**
      Apply a 3x3 blur filter: Replace each pixel with the average of its neighbors (including itself). If a pixel is at the edge or corner, consider only valid neighbors.
   2. **Sharpen (Code: 2)**
      Apply a 3x3 sharpening kernel:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

3. **Emboss (Code: 3)**
   Apply an emboss filter using the following kernel:

$$\begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

Input Format

- The first input is an integer n ($1 \leq n \leq 100$) representing the size of the square matrix.
- The next n lines contain n integers each ($0 \leq$ value ≤ 255), representing the grayscale image matrix.
- The final input is an integer code (1, 2, or 3), indicating the filter to apply.

Output Format

Output the transformed matrix after applying the filter. Each element should be an integer and must be clamped between 0 and 255.

Notes

- Implement edge handling such that pixels on the edge/corner are processed with only valid neighbors.
- Ensure that the resulting matrix values are clamped to the range [0, 255].

Example with Mathematical Calculations

Consider a small 3×3 grayscale image matrix, and apply the **Sharpen filter (Code: 3)**:

Input

$$\text{Image Matrix: } \begin{bmatrix} 50 & 80 & 50 \\ 90 & 100 & 90 \\ 50 & 80 & 50 \end{bmatrix}$$

$$\text{Sharpen Kernel: } \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Step-by-Step Calculation

For each pixel, compute the weighted sum of the pixel and its neighbors using the kernel. Pixels at the edges or corners use only valid neighbors.

Example Calculation for the Center Pixel (100):

$$\text{Neighborhood: } \begin{bmatrix} 50 & 80 & 50 \\ 90 & 100 & 90 \\ 50 & 80 & 50 \end{bmatrix}$$

Apply the kernel:

$$(0\cdot 50)+(-1\cdot 80)+(0\cdot 50)+(-1\cdot 90)+(5\cdot 100)+(-1\cdot 90)+(0\cdot 50)+(-1\cdot 80)+(0\cdot 50)$$

$$= 0 - 80 + 0 - 90 + 500 - 90 + 0 - 80 + 0$$

$$= 160$$

Clamp the result to $[0, 255]$, so the pixel value remains 160.

Calculations for Other Pixels:

Perform similar calculations for each pixel. For edge or corner pixels, consider only valid neighbors.

Test Cases:

1. Box Blur (Code: 1)

Input

```
3
10 20 30
40 50 60
70 80 90
1
```

Output

```
30 35 40
45 50 55
60 65 70
```

—

2. Sharpen (Code: 2)

Input

```
5
10 10 10 10 10
10 20 20 20 10
10 20 50 20 10
10 20 20 20 10
10 10 10 10 10
2
```

Output

```
30  10  10  10  30
10  40   0  40  10
10   0 170   0  10
10  40   0  40  10
30  10  10  10  30
```

—

3. Emboss (Code: 3)

Input

```
3
1 1 1
1 1 1
1 1 1
3
```

Output

```
5 4 1
4 1 0
0 0 0
```

(6 marks)

2. In quantum mechanics, a quantum gate is represented as a matrix that operates on a quantum state vector to produce a new state. Write a program that takes a quantum gate (matrix) and a quantum state vector as input and calculates the resulting state vector after applying the gate operation.

Input

1. An integer n representing the size of the quantum gate and state vector (i.e., $n \times n$ gate and n-dimensional vector).
2. An $n \times 1$ real-valued matrix representing the quantum gate.
3. An $n \times 1$ real-valued vector representing the initial quantum state.

Output

- An n-dimensional vector representing the final quantum state after the gate operation.

Example 1

Input

```
2
0 1
1 0
1 0
```

Output

```
0 1
```

Example 2

Input

```
3
1 0 0
0 0 1
0 1 0
1 2 3
```

Output

```
1 3 2
```

Explanation of Example 1

For $n = 2$, the input gate and state vector are:

$$\text{Gate: } \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \text{State Vector: } \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

The resulting state vector is computed as:

$$\text{Final State: } \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Explanation of Example 2

For $n = 3$, the input gate and state vector are:

$$\text{Gate: } \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad \text{State Vector: } \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}.$$

The resulting state vector is computed as:

$$\text{Final State: } \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}.$$

Test Cases

Test Case 1

Input:

```
2
1 0
0 1
5 7
```

Output:

```
5 7
```

Test Case 2

Input:

```
2
0 1
1 0
4 9
```

Output:

```
9 4
```

Test Case 3

Input:

```
3
0 1 0
0 0 1
1 0 0
8 6 4
```

Output:

```
6 4 8
```

Test Case 4

Input:

```
3
1 0 0
0 0 1
0 1 0
2 5 3
```

Output:

```
2 3 5
```

(4 marks)

All the Best!