CHRIST
(DEEMED TO BE UNIVERSITY)
BENGALURU · INDIA

DIGITAL PATH FINDING AND REGION DETECTION IN BINARY IMAGES

SANJAY R
2348055

7th October 2024

MDS573C
Image And Video Analytics
Department of Statistics and Data Science

CHRIST (Deemed to be University)

Department of Statistics and Data Science

Course: MDS573C - Image And Video Analytics **Lab :1**

R. SANJAY **2348055**

Digital Path finding and Region Detection in Binary Images

✧ *Introduction:*

In the realm of digital image processing, understanding pixel connectivity and identifying distinct regions within an image are fundamental components of effective image analysis. As images are essentially composed of a grid of pixels, each with an associated intensity value, analyzing the spatial relationships between these pixels becomes crucial in various applications, including computer vision, object detection, medical imaging, and pattern recognition.

Connectivity defines how pixels relate to one another based on their adjacency, significantly influencing how we interpret and manipulate images. The classification of pixels into connected groups helps in identifying objects, edges, and areas of interest. In this report, we will explore two essential tasks that delve into pixel connectivity and region identification:

- **Task 1:** Finding digital paths between pixels using three distinct types of adjacency rules: 4-path, 8-path, and m-path. Each of these paths offers different ways of connecting pixels based on their arrangement. The 4-path focuses solely on horizontal and vertical connections, while the 8-path expands this by including diagonal connections. The m-path combines elements of both but imposes additional constraints on diagonal connectivity. Understanding these paths enables us to traverse images more effectively, allowing for accurate navigation through pixel arrangements and facilitating tasks such as segmentation and feature extraction.

- **Task 2:** Identifying distinct regions in a binary image and determining whether these regions are adjacent or disjoint. Regions are formed by groups of connected pixels that share the same intensity value (in this case, 1). Analyzing the adjacency of these regions is crucial for understanding the structure of an image and can significantly impact further processing tasks. For instance, adjacent regions may indicate potential overlaps or relationships between objects in an image, while disjoint regions could represent separate entities.

The culmination of these tasks provides a comprehensive framework for analyzing images at both the micro and macro levels. By understanding pixel connectivity and the formation of regions, we can leverage these insights to enhance various image processing techniques. Whether in autonomous vehicles for obstacle detection, in medical imaging for tumor identification, or in social media for image recognition, mastering pixel adjacency and region identification is pivotal for achieving accurate and reliable outcomes.Through this report, we aim to illustrate the importance of pixel paths and region connectivity in digital images, providing a foundation for further exploration into more complex image processing algorithms and methodologies.

✧ *Description of the Exercise:*

The exercise focuses on implementing digital pathfinding and region detection in a 5x5 binary image using different connectivity rules: 4-path, 8-path, and m-path. The binary image is composed of pixels with intensity values `0` and `1`, where `1` represents foreground pixels of interest. The goal is to find all possible paths between any two foreground pixels (`p` to `q`) based on specified adjacency criteria.

In the first part of the exercise, paths are determined using 4-path connectivity, which only considers horizontal and vertical neighbors, 8-path connectivity, which includes diagonal neighbors as well, and m-path connectivity, which combines the two by applying certain conditions. These different pathfinding methods are used to compute and print all possible traversals between any two foreground pixels, highlighting their accessibility based on adjacency rules.

The second part of the exercise involves detecting regions in the binary image. A region is defined as a connected set of foreground pixels. Using depth-first search (DFS), the image is analyzed to find all such regions, and then the regions are classified as either adjacent or disjoint based on their proximity to one another. The adjacency is determined using 8-path connectivity, which checks if any pixels from different regions are neighbors. The task demonstrates how to analyze spatial relationships and connected components in binary images without the use of external libraries, making it a fundamental exercise in image and video analytics.

✧ *Exercise question:*

Assume a binary image size 5 x 5 and perform the following:
Let V ={1} be the set of intensity value to define the adjacency.

(a) Find the following digital paths and print the path traversal from any source pixel 'p' to any other source pixel 'q' 4-Path, 8-Path and m-Path

The task involves determining paths between any source pixel p and all possible destination pixels q in a 5x5 binary image using three pathfinding strategies: 4-Path, 8-Path, and m-Path. In this binary image, 1 represents the foreground (white) pixels, while 0 represents the background (black) pixels. The set V={1} defines pixel adjacency for path traversal. Using 4-Path, only horizontal and vertical neighbors are considered, while 8-Path includes diagonal neighbors as well. m-Path combines both, with specific conditions for connectivity. The task aims to find and print all possible paths between pairs of 1s using these adjacency rules.

✧ Concepts Used:

1. 4-Path Adjacency:

In 4-path adjacency, two pixels are considered connected if they share a common edge. This means that only the pixels immediately above, below, to the left, or to the right of a given pixel are regarded as its neighbors. This connectivity is particularly useful for traversing pixel arrangements in a grid layout.

For a pixel located at coordinates (x,y), the 4-path neighbors are:
$(x-1,y)$ (above) , $(x+1,y)$ (below),$(x,y-1)$ (left), $(x,y+1)$ (right)

2. 8-Path Adjacency:

In 8-path adjacency, two pixels are considered connected if they share a common edge or corner. This means that a pixel can connect to its immediate horizontal, vertical, and diagonal neighbors, providing a more comprehensive way to traverse pixel arrangements.

For a pixel at (x,y), the 8-path neighbors include:

Horizontal/Vertical neighbors: $(x-1,y)$, $(x+1,y)$, $(x,y-1)$, $(x,y+1)$
Diagonal neighbors: $(x-1,y-1)$,$(x-1,y+1)$,$(x+1,y-1)$, $(x+1,y+1)$

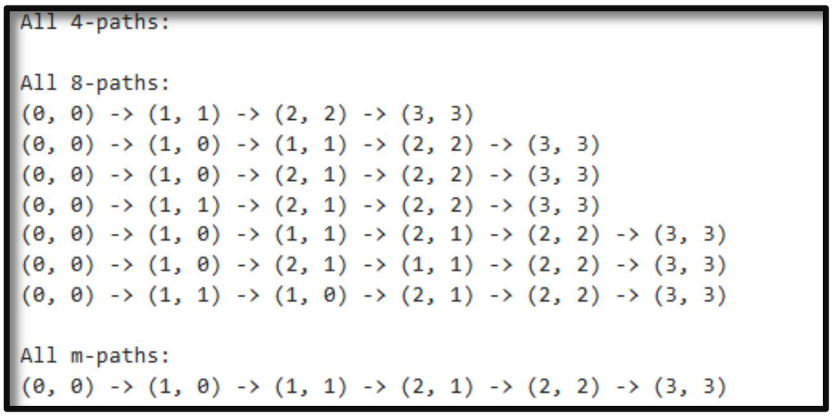
3. m-Path Adjacency:

The m-path adjacency model allows diagonal connections between pixels but only under specific conditions. A diagonal connection is permitted if the immediate horizontal and vertical neighbors of the pixels involved are not connected. This

model provides a balance between the strict 4-path and the more lenient 8-path connections.
For a pixel at (x,y), a diagonal connection to (x−1,y−1) is only valid if:
The pixel above (x−1,y) and the pixel to the left (x,y−1) are not connected.

✧ *Python Code:*

```
import numpy as np
from collections import deque

image = np.array([ # Sample Binary image(5*5 Matrix)
    [1, 0, 0, 0, 1],
    [1, 1, 0, 0, 1],
    [0, 1, 1, 0, 0],
    [0, 0, 0, 1, 1],
    [1, 1, 0, 0, 0]
])
start_pixel = (0, 0)  # Start at top-left
end_pixel = (3, 3)    # End at (3, 3)
v={1}

def find_all_paths_4(image, start, end):
    directions_4 = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # 4-path directions
      return bfs_all(image, start, end, directions_4)

def find_all_paths_8(image, start, end):
    directions_8 = [(-1, 0), (1, 0), (0, -1), (0, 1), (-1, -1), (-1, 1), (1, -1), (1, 1)]  # 8-path directions
      return bfs_all(image, start, end, directions_8)

def find_all_paths_m(image, start, end):
    directions_4 = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # 4-path directions
    directions_8 = [(-1, -1), (-1, 1), (1, -1), (1, 1)]  # Diagonal directions (8-path)
     return bfs_all_m(image, start, end, directions_4, directions_8)

def bfs_all(image, start, end, directions): # BFS implementation to find all paths
    rows, cols = image.shape
    queue = deque([(start, [start])])  # Queue stores (current node, current path)
    all_paths = []  # To collect all paths

    while queue:
        current, path = queue.popleft()

        # If we reached the end, save the path
        if current == end:
            all_paths.append(path)
            continue  # Continue to explore other paths

        # Explore neighbors
        for direction in directions:
            nr, nc = current[0] + direction[0], current[1] + direction[1]
            if (0 <= nr < rows and 0 <= nc < cols and
                image[nr, nc] == 1 and (nr, nc) not in path):
                # Add new path to the queue
                queue.append(((nr, nc), path + [(nr, nc)]))

     return all_paths  # Return all paths found
     def bfs_all_m(image, start, end, directions_4, directions_8): # BFS with priority to 4-path first, then 8-path
(diagonal) moves
         rows, cols = image.shape
         queue = deque([(start, [start])])  # Queue stores (current node, current path)
         all_paths = []  # To collect all paths

         while queue:
             current, path = queue.popleft()
            # If we reached the end, save the path
            if current == end:
                all_paths.append(path)
                continue  # Continue to explore other paths

            # Check for 4-path neighbors
            neighbors_found = False  # Track if any 4-path neighbors are found
            for direction in directions_4:
                nr, nc = current[0] + direction[0], current[1] + direction[1]
                if (0 <= nr < rows and 0 <= nc < cols and
                    image[nr, nc] == 1 and (nr, nc) not in path):
                    queue.append(((nr, nc), path + [(nr, nc)]))  # Continue with 4-path
                    neighbors_found = True  # Mark that we found 4-path neighbors

            # Only check 8-path neighbors if no 4-path neighbors were found
            if not neighbors_found:
                for direction in directions_8:
                    nr, nc = current[0] + direction[0], current[1] + direction[1]
                    if (0 <= nr < rows and 0 <= nc < cols and
                        image[nr, nc] == 1 and (nr, nc) not in path):
                        queue.append(((nr, nc), path + [(nr, nc)]))  # Continue exploring diagonals

        return all_paths  # Return all paths found

# Find and print all 4-paths, 8-paths, and m-paths
all_paths_4 = find_all_paths_4(image, start_pixel, end_pixel)
print("All 4-paths:")
for path in all_paths_4:
    print(" -> ".join(str(p) for p in path))

all_paths_8 = find_all_paths_8(image, start_pixel, end_pixel)
print("\nAll 8-paths:")
for path in all_paths_8:
    print(" -> ".join(str(p) for p in path))

all_paths_m = find_all_paths_m(image, start_pixel, end_pixel)
print("\nAll m-paths:")
for path in all_paths_m:
     print(" -> ".join(str(p) for p in path))
```



```
All 4-paths:

All 8-paths:
(0, 0) -> (1, 1) -> (2, 2) -> (3, 3)
(0, 0) -> (1, 0) -> (1, 1) -> (2, 2) -> (3, 3)
(0, 0) -> (1, 0) -> (2, 1) -> (2, 2) -> (3, 3)
(0, 0) -> (1, 1) -> (2, 1) -> (2, 2) -> (3, 3)
(0, 0) -> (1, 0) -> (1, 1) -> (2, 1) -> (2, 2) -> (3, 3)
(0, 0) -> (1, 0) -> (2, 1) -> (1, 1) -> (2, 2) -> (3, 3)
(0, 0) -> (1, 1) -> (1, 0) -> (2, 1) -> (2, 2) -> (3, 3)

All m-paths:
(0, 0) -> (1, 0) -> (1, 1) -> (2, 1) -> (2, 2) -> (3, 3)
```

(b) Find all the regions present in the image and declare the adjacent regions and disjoint regions.

Region: A region in a binary image is defined as a connected set of pixels that share the same intensity value. In our case, a region consists of pixels with the value `1`, which are considered "on" pixels. The connectedness can be defined based on the pixel adjacency rules (4-path, 8-path, or m-path).

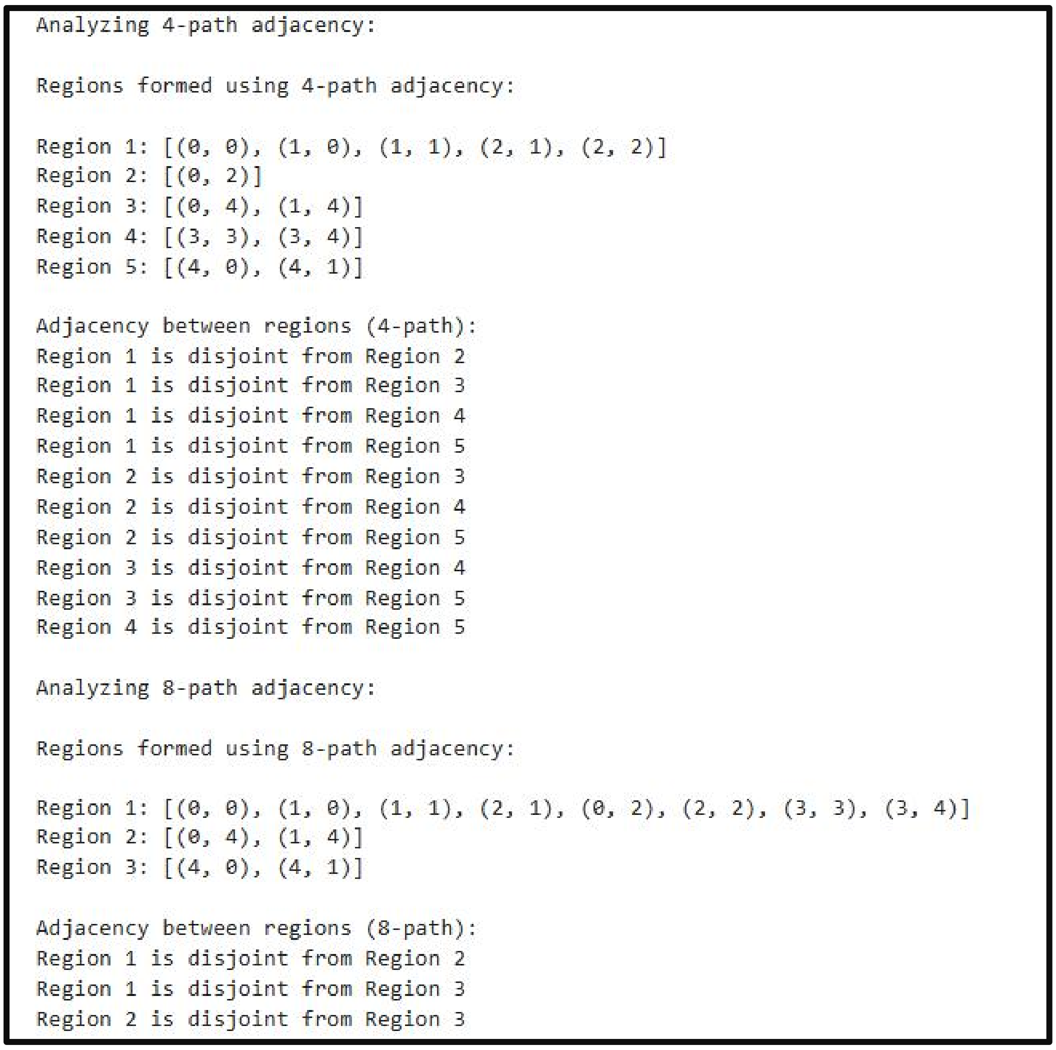
Adjacent Regions: Adjacent regions are defined as regions that are connected to each other either directly or indirectly through a shared edge or corner. In an 8-path adjacency, this includes diagonal connections.

Disjoint Regions: Disjoint regions are regions that do not share any pixels or connections. They are completely separate from one another in the pixel grid.

✧ *Python Code:*

```
def label_regions(image, directions):
    rows, cols = image.shape
    labeled_image = np.zeros_like(image)  # To store labeled regions
    label = 1  # Starting label
    regions = {}  # Dictionary to hold regions

    def bfs(start):
        queue = deque([start])
        region = []
        labeled_image[start] = label
        region.append(start)

        while queue:
            current = queue.popleft()
            for direction in directions:
                nr, nc = current[0] + direction[0], current[1] + direction[1]
                if 0 <= nr < rows and 0 <= nc < cols and image[nr, nc] == 1 and labeled_image[nr, nc] == 0:
                    queue.append((nr, nc))
                    labeled_image[nr, nc] = label
                    region.append((nr, nc))

        return region

    # Find all regions
    for r in range(rows):
        for c in range(cols):
            if image[r, c] == 1 and labeled_image[r, c] == 0:
                regions[label] = bfs((r, c))
                label += 1

    return labeled_image, regions

# Function to check if two regions are adjacent
def are_adjacent(region1, region2, directions):
    for r1, c1 in region1:
        for direction in directions:
            nr, nc = r1 + direction[0], c1 + direction[1]
            if (nr, nc) in region2:
                return True
    return False
```

```
def analyze_regions(image, directions, adjacency_type="4-path"):
    labeled_image, regions = label_regions(image, directions)

    print(f"\nRegions formed using {adjacency_type} adjacency:\n")
    for label, region in regions.items():
        print(f"Region {label}: {region}")

    # Checking adjacency between regions
    print(f"\nAdjacency between regions ({adjacency_type}):")
    region_labels = list(regions.keys())
    for i in range(len(region_labels)):
        for j in range(i + 1, len(region_labels)):
            region1 = regions[region_labels[i]]
            region2 = regions[region_labels[j]]
            if are_adjacent(region1, region2, directions):
                print(f"Region {region_labels[i]} is adjacent to Region {region_labels[j]}")
            else:
                print(f"Region {region_labels[i]} is disjoint from Region {region_labels[j]}")

# Run the analysis for both 4-path and 8-path
print("Analyzing 4-path adjacency:")
analyze_regions(image, directions_4, "4-path")

print("\nAnalyzing 8-path adjacency:")
analyze_regions(image, directions_8, "8-path")
```



```
Analyzing 4-path adjacency:

Regions formed using 4-path adjacency:

Region 1: [(0, 0), (1, 0), (1, 1), (2, 1), (2, 2)]
Region 2: [(0, 2)]
Region 3: [(0, 4), (1, 4)]
Region 4: [(3, 3), (3, 4)]
Region 5: [(4, 0), (4, 1)]

Adjacency between regions (4-path):
Region 1 is disjoint from Region 2
Region 1 is disjoint from Region 3
Region 1 is disjoint from Region 4
Region 1 is disjoint from Region 5
Region 2 is disjoint from Region 3
Region 2 is disjoint from Region 4
Region 2 is disjoint from Region 5
Region 3 is disjoint from Region 4
Region 3 is disjoint from Region 5
Region 4 is disjoint from Region 5

Analyzing 8-path adjacency:

Regions formed using 8-path adjacency:

Region 1: [(0, 0), (1, 0), (1, 1), (2, 1), (0, 2), (2, 2), (3, 3), (3, 4)]
Region 2: [(0, 4), (1, 4)]
Region 3: [(4, 0), (4, 1)]

Adjacency between regions (8-path):
Region 1 is disjoint from Region 2
Region 1 is disjoint from Region 3
Region 2 is disjoint from Region 3
```

✧ *Conclusion:*

In this report, we have thoroughly explored the critical aspects of pixel connectivity and region identification within digital images. Through Task 1, we examined how different types of digital paths—4-path, 8-path, and m-path—provide distinct methods for traversing images based on pixel adjacency. Each path offers unique advantages in determining how pixels are connected, which plays a vital role in various image processing applications. For instance, understanding these paths can facilitate the segmentation of images, allowing for precise feature extraction and enabling algorithms to discern patterns and objects more effectively.

Task 2 built upon this foundation by identifying distinct regions in a binary image and analyzing the adjacency and disjoint characteristics of these regions. By leveraging pixel connectivity, we established a framework for understanding how pixels group together to form larger structures within an image. This analysis is particularly important in applications such as object detection, where identifying and differentiating between overlapping regions can significantly affect the performance of machine learning models and image classification systems.

The insights gained from examining pixel paths and region adjacency are not just theoretical; they have practical implications across various domains. In medical imaging, for example, accurate identification of connected regions can aid in diagnosing conditions based on the shape and size of tumors. In autonomous systems, understanding the spatial relationships between objects can enhance navigation and obstacle detection capabilities. Additionally, in the realm of computer vision, these techniques are foundational for developing sophisticated algorithms that drive advancements in areas like augmented reality and facial recognition.

In summary, this report underscores the importance of pixel connectivity and region identification in digital image processing. By mastering these concepts, we lay the groundwork for further exploration into advanced techniques, paving the way for innovative applications that harness the power of image analysis. The ability to accurately traverse images and identify significant regions is pivotal in transforming raw image data into meaningful insights, ultimately enhancing our interaction with the visual world.