CSC311 Project

Seam Carving - Phase 2

(Greedy and Dynamic approach)

GROUP :

#9

SECTION:

44078

Sara Alshddi	Nada almalq	Sultanah Alahmed
444200989	444200899	444203969

TABLE OF CONTENTS

GREEDY APPROACH

```
Algorithm GreedySeamCarving(image, numSeams):
   // Input: image to process, number of seams to remove
   // Output: resized image after removing seams

   for i from 1 to numSeams do:
      energyMatrix ← ComputeEnergy(image)
      // Calculate energy of each pixel using Sobel operator (gradient magnitude)

      seam ← FindGreedySeam(energyMatrix)
      // Select the seam by choosing locally minimum energy at each step (greedy)

      image ← RemoveSeam(image, seam)
      // Remove the selected seam from the image
   end for

   return image

Algorithm ComputeEnergy(image):
   // Input: RGB image
   // Output: 2D array representing energy of each pixel

   Convert image to grayscale
   // Average R, G, B values for each pixel

   Apply Sobel filter in X and Y directions
   // Calculate gradients to detect edges

   For each pixel:
      energy ← sqrt(gx^2 + gy^2)
      // Energy is the magnitude of the gradient

   return energyMatrix


Algorithm FindGreedySeam(energyMatrix):
   // Input: energy matrix
   // Output: array representing the seam path (column indices per row)

   Find the pixel with minimum energy in the first row
   // Start seam from the top row, least energy pixel

   For each row from top to bottom:
      Move to the neighbor pixel (left, center, or right) with the lowest energy
      // At each step, only consider immediate neighbors (greedy choice)

   return seamPath


Algorithm RemoveSeam(image, seam):
   // Input: image and seam path
   // Output: new image with one seam removed

   Create a new image with width - 1

   For each row:
      Copy pixels from old image to new image, skipping the seam pixel
      // Shift pixels after the seam to the left

   return newImage
```

DYNAMIC APPROACH

```
Algorithm DynamicSeamCarving(image, numSeams):
   // Input: image to process, number of seams to remove
   // Output: resized image after removing seams

   for i from 1 to numSeams do:
      energyMatrix ← ComputeEnergy(image)
      // Calculate energy of each pixel

      cumulativeEnergy, backtrack ← ComputeCumulativeEnergy(energyMatrix)
      // Build cumulative minimum energy map (dynamic programming)
      // backtrack matrix saves the path to reconstruct the seam

      seam ← FindSeamDP(cumulativeEnergy, backtrack)
      // Recover the minimum energy seam path by backtracking

      image ← RemoveSeam(image, seam)
      // Remove the selected seam from the image
   end for

   return image

Algorithm ComputeCumulativeEnergy(energyMatrix):
   // Input: energy matrix
   // Output: cumulative energy matrix and backtrack matrix

   Initialize cumulativeEnergy matrix as a copy of energyMatrix
   Initialize backtrack matrix to store paths

   For each row from second row to bottom:
      For each pixel:
         Find the minimum cumulative energy among the three possible top neighbors (left, center, right)
         Update cumulativeEnergy for current pixel:
            cumulativeEnergy[y][x] ← energy[y][x] + minimum(cumulativeEnergy[y-1][x-1], cumulativeEnergy[y-1][x],
cumulativeEnergy[y-1][x+1])
         Update backtrack[y][x] with the column index of the chosen neighbor
         // Save which direction we came from to allow path reconstruction

   return cumulativeEnergy, backtrack


Algorithm FindSeamDP(cumulativeEnergy, backtrack):
   // Input: cumulative energy matrix and backtrack matrix
   // Output: seam path

   Find the column index with the minimum energy in the last row
   // Start from the bottom row, least cumulative energy

   For each row from bottom to top:
      Add current column index to seam path
      Move to the parent pixel based on backtrack matrix
      // Trace back the minimum energy path step by step

   return seamPath


Algorithm RemoveSeam(image, seam):
   // Input: image and seam path
   // Output: new image with one seam removed

   Create a new image with width - 1

   For each row:
      Copy pixels from old image to new image, skipping the seam pixel
      // Shift pixels after the seam to the left

   return newImage
```

Comparison Between Dynamic and Greedy Algorithms

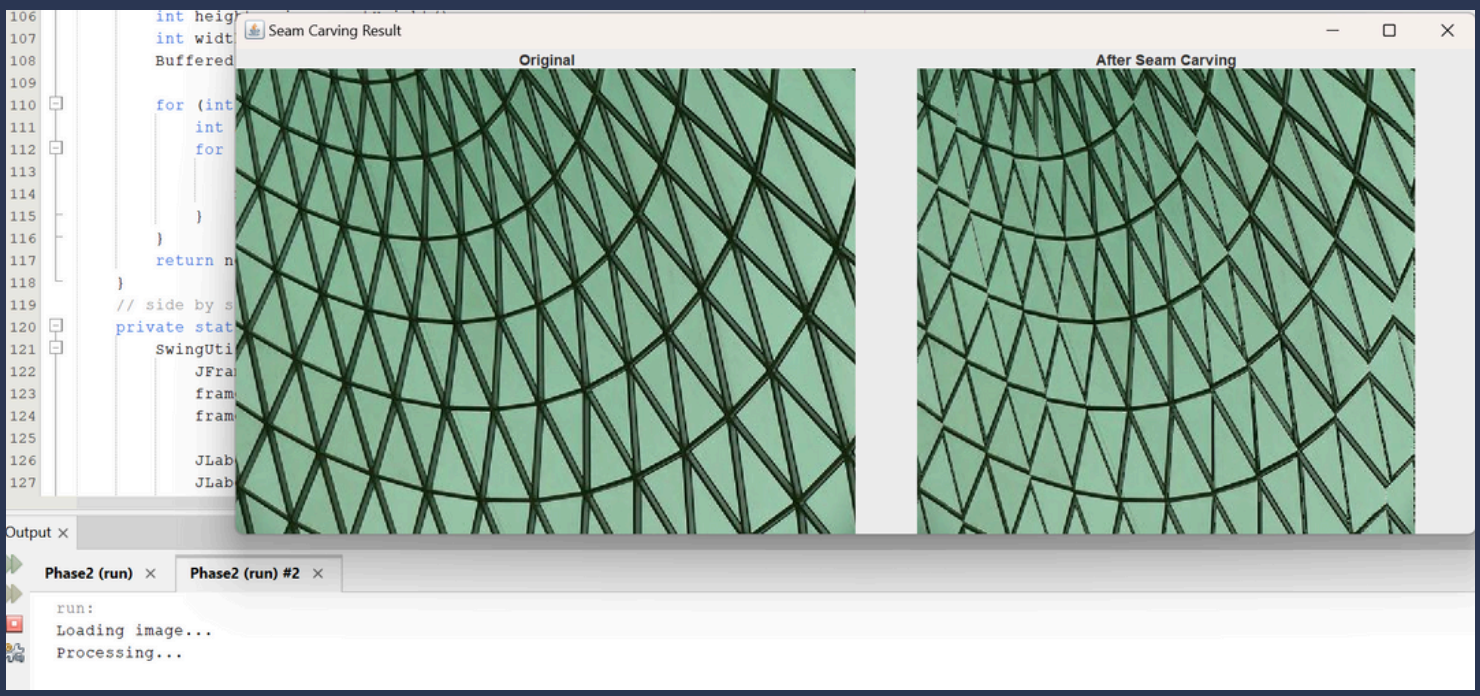
Aspect	Dynamic Programming Algorithm	Greedy Algorithm
Method of Calculation	Uses dynamic programming to compute cumulative energy (bottom-up) and finds the optimal seam path.	Selects the pixel with the least energy at each step and moves to the next row.
Time Complexity	O(w * h) for each pixel (where w is width and h is height).	O(w * h) for each pixel, but does not involve deep calculations.
Accuracy	More accurate as it computes the cumulative energy and searches for the optimal path.	May miss the optimal path due to reliance on local choices.
Speed	Slower as it calculates cumulative energy for all pixels.	Faster because the algorithm selects pixels quickly using the greedy approach.
Space Complexity	Requires storing multiple arrays (cumulative energy array and traceback array).	Less space complexity as it uses only one array for energy.
Suitability	Better for images with more complexity (e.g., areas with high contrast).	Suitable for simpler scenes but might fail on complex ones.
Stability in Performance	Stable and consistent in results.	Less stable and may produce suboptimal results on complex images.
Edge Case Handling	Handles edge cases better as it explores multiple options.	Can struggle with edge cases due to the greedy choice property.
Advantages	- Produces better quality results in complex scenarios. - Optimal solution based on global analysis.	- Faster processing time. - Lower memory usage.
Disadvantages	- Slower processing due to complex calculations. - Higher memory usage.	- May result in suboptimal solutions. - Doesn't guarantee the best outcome.

Time Complexities Table

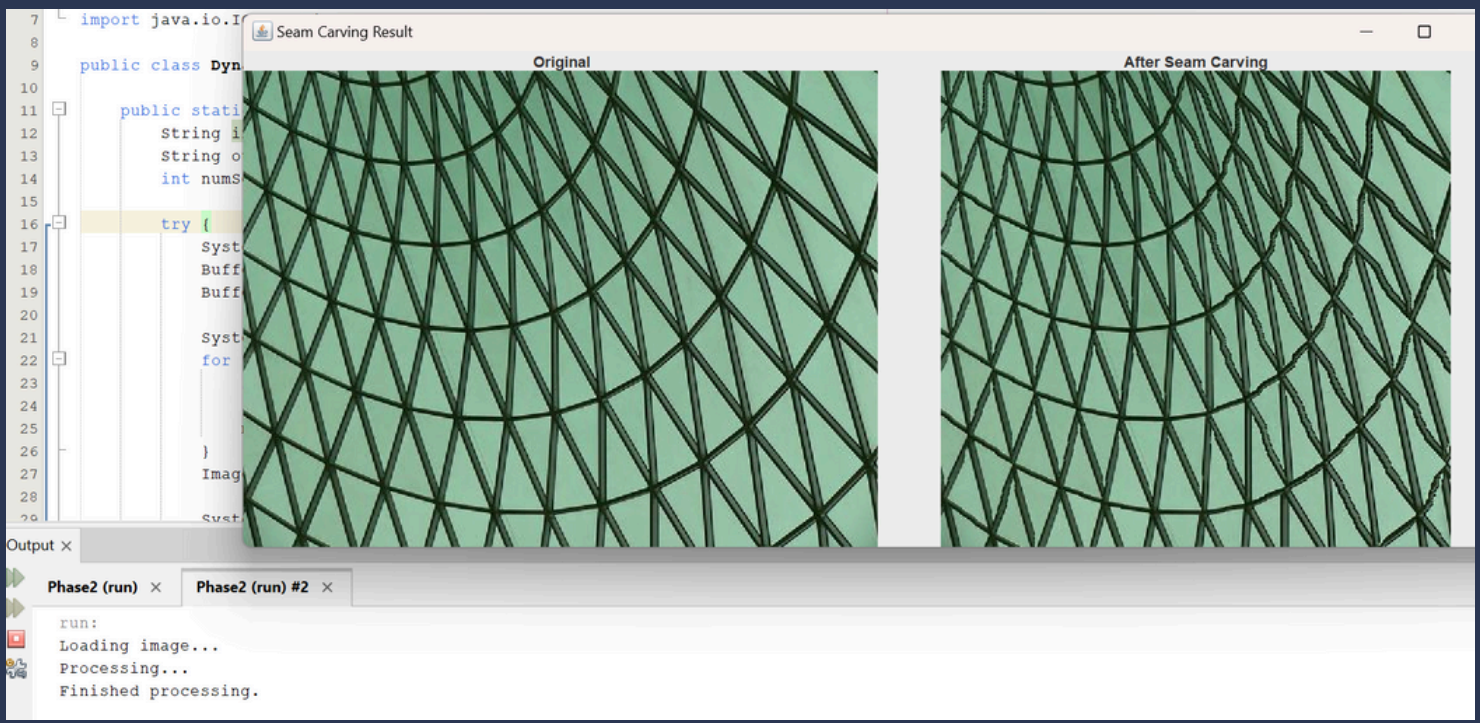
Algorithm	Best Case Complexity	Example (Best Case)	Worst Case Complexity	Example (Worst Case)
Greedy	O(height)	A simple image with a straight vertical low-energy path (like a vertical black line on white background).	O(height × width)	A noisy image with random textures where every step needs checking left, middle, and right.
Dynamic Programming	O(height × width)	A uniform image (all white), where all paths have the same energy, leading to easy cumulative calculations.	O(height × width)	A detailed natural image (like a forest with trees and shadows) where every pixel energy matters.
Brute Force	O(width × 3^height) (small image)	A very small image (e.g., 3×3 or 5×5), few possible paths, recursion ends fast.	O(width × 3^height)	A large complex image (e.g., 500×500) where the algorithm has to check every possible seam recursively.

SAMPLE RUN

Greedy Algorithm



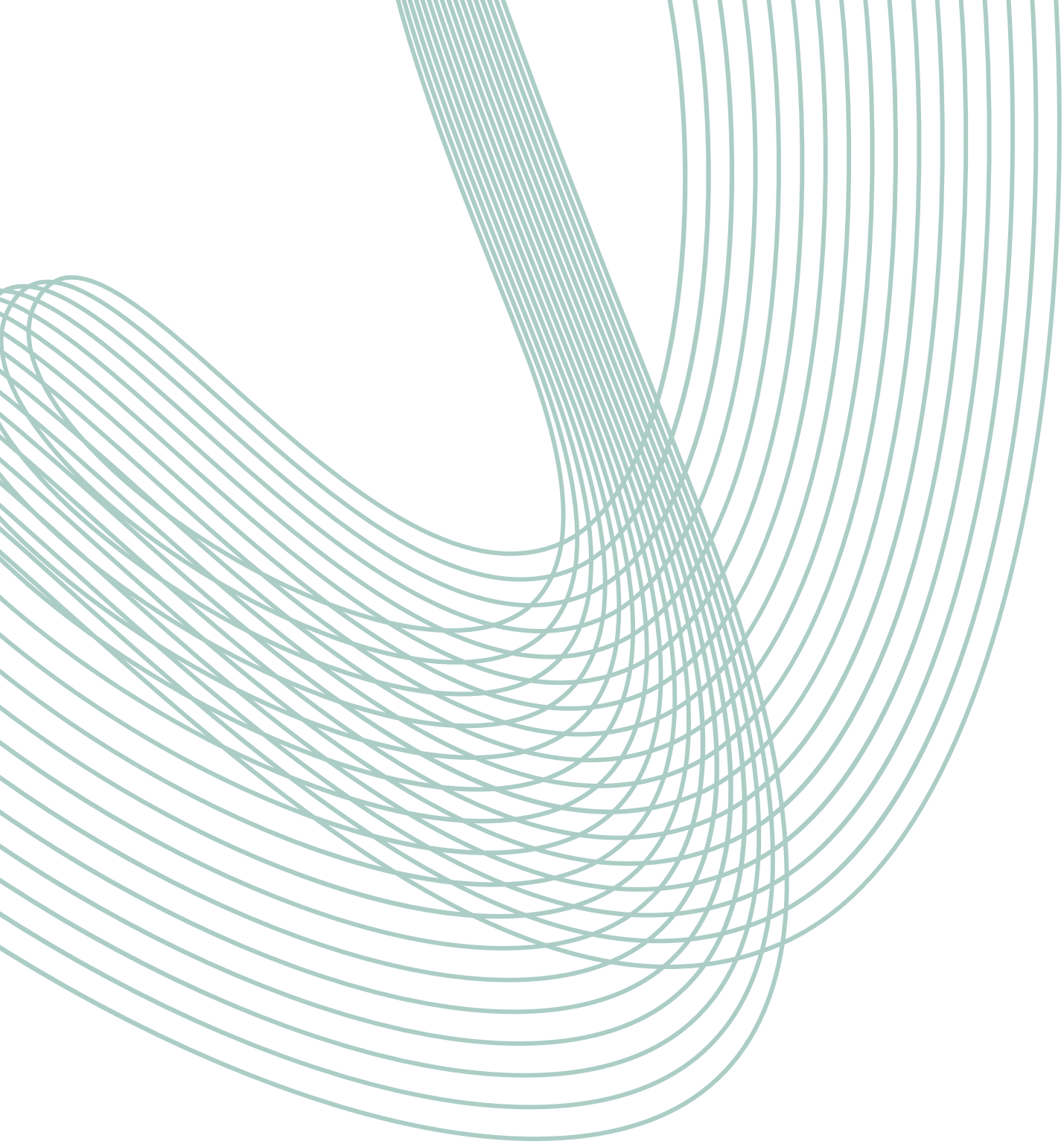
Dynamic Programming



GREEDY ALGORITHM SOURCE CODE

```
import java.awt.*;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import javax.swing.*;
import java.io.File;
import java.io.IOException;

public class Greedy {
    public static void main(String[] args) {
    String inputImagePath = "C:\\Users\\honey\\Desktop\\images\\museum.jpg"; // Path to input image
    String outputImagePath = "C:\\Users\\honey\\Desktop\\images\\greedy_result.jpg"; // Path to save output
        int numSeams = 100; // Number of seams to remove

        try {
            BufferedImage originalImage = ImageIO.read(new File(inputImagePath));
            BufferedImage modifiedImage = originalImage;

            for (int i = 0; i < numSeams; i++) {
                int[][] energy = computeEnergy(modifiedImage); // Compute energy map
                int[] seam = findSeamGreedy(energy); // Find seam using greedy approach
                modifiedImage = removeSeam(modifiedImage, seam); // Remove the seam
            }

            ImageIO.write(modifiedImage, "jpg", new File(outputImagePath));
            showSideBySide(originalImage, modifiedImage); // Display original and modified images
        } catch (IOException e) {
            System.err.println("Error: " + e.getMessage());
        }
    }

    private static int[][] computeEnergy(BufferedImage image) {
        int height = image.getHeight();
        int width = image.getWidth();
        int[][] gray = new int[height][width];
        int[][] energy = new int[height][width];

        // Convert to grayscale
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                int color = image.getRGB(x, y);
                int r = (color >> 16) & 0xff;
                int g = (color >> 8) & 0xff;
                int b = color & 0xff;
                gray[y][x] = (r + g + b) / 3;
            }
        }

        int[][] sobelX = { {-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1} };
        int[][] sobelY = { {1, 2, 1}, {0, 0, 0}, {-1, -2, -1} };

        // Apply Sobel operator
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                int gx = 0, gy = 0;
                for (int i = -1; i <= 1; i++) {
                    for (int j = -1; j <= 1; j++) {
                        int ny = Math.min(Math.max(y + i, 0), height - 1);
                        int nx = Math.min(Math.max(x + j, 0), width - 1);
                        gx += sobelX[i + 1][j + 1] * gray[ny][nx];
                        gy += sobelY[i + 1][j + 1] * gray[ny][nx];
                    }
                }
                energy[y][x] = (int) Math.sqrt(gx * gx + gy * gy);
            }
        }
        return energy;
    }

    private static int[] findSeamGreedy(int[][] energy) {
        int height = energy.length;
        int width = energy[0].length;
        int[] seam = new int[height];

        // Start with the lowest energy pixel in the first row
        int minIndex = 0;
        for (int j = 1; j < width; j++) {
            if (energy[0][j] < energy[0][minIndex]) {
                minIndex = j;
            }
        }
        seam[0] = minIndex;

        // Move greedily downwards
        for (int i = 1; i < height; i++) {
            int prevX = seam[i - 1];
            int newX = prevX;
            int minEnergy = energy[i][prevX];
```

```
            if (prevX > 0 && energy[i][prevX - 1] < minEnergy) {
                newX = prevX - 1;
                minEnergy = energy[i][prevX - 1];
            }
            if (prevX < width - 1 && energy[i][prevX + 1] < minEnergy) {
                newX = prevX + 1;
            }
            seam[i] = newX;
        }
        return seam;
    }

    private static BufferedImage removeSeam(BufferedImage image, int[] seam) {
        int height = image.getHeight();
        int width = image.getWidth();
        BufferedImage newImage = new BufferedImage(width - 1, height, BufferedImage.TYPE_INT_RGB);

        for (int y = 0; y < height; y++) {
            int seamX = seam[y];
            for (int x = 0; x < width - 1; x++) {
                int srcX = (x < seamX) ? x : x + 1;
                newImage.setRGB(x, y, image.getRGB(srcX, y));
            }
        }
        return newImage;
    }

    private static void showSideBySide(BufferedImage before, BufferedImage after) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("Seam Carving Result");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setLayout(new GridLayout(1, 2));

            JLabel beforeLabel = new JLabel(new ImageIcon(before));
            JLabel afterLabel = new JLabel(new ImageIcon(after));

            JPanel beforePanel = new JPanel(new BorderLayout());
            beforePanel.add(new JLabel("Original", SwingConstants.CENTER), BorderLayout.NORTH);
            beforePanel.add(beforeLabel, BorderLayout.CENTER);

            JPanel afterPanel = new JPanel(new BorderLayout());
            afterPanel.add(new JLabel("After Seam Carving", SwingConstants.CENTER), BorderLayout.NORTH);
            afterPanel.add(afterLabel, BorderLayout.CENTER);

            frame.add(beforePanel);
            frame.add(afterPanel);
            frame.pack();
            frame.setLocationRelativeTo(null);
            frame.setVisible(true);
        });
    }
}
```



DYNAMIC PROGRAMMING SOURCE CODE

```
import java.awt.*;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import java.io.File;
import java.io.IOException;
import javax.swing.*;

public class Dynamic {

    public static void main(String[] args) {
        String inputImagePath = "C:\\Users\\honey\\Desktop\\images\\museum.jpg"; // Input image path
          String outputImagePath = "C:\\Users\\honey\\Desktop\\images\\dynamic_result.jpg"; // Output image
path
        int numSeams = 100; // Number of seams to remove

        try {
            BufferedImage originalImage = ImageIO.read(new File(inputImagePath));
            BufferedImage modifiedImage = originalImage;

            for (int i = 0; i < numSeams; i++) {
                double[][] energy = computeEnergy(modifiedImage); // Compute energy
                int[] seam = findSeamDp(energy); // Find optimal seam using DP
                modifiedImage = removeSeam(modifiedImage, seam); // Remove seam
            }

            ImageIO.write(modifiedImage, "jpg", new File(outputImagePath));
            showSideBySide(originalImage, modifiedImage); // Display results
        } catch (IOException e) {
            System.err.println("Error: " + e.getMessage());
        }
    }

    private static double[][] computeEnergy(BufferedImage image) {
        int width = image.getWidth();
        int height = image.getHeight();
        double[][] energy = new double[height][width];

        // Compute brightness gradient (simple method)
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                int color = image.getRGB(x, y);
                int r = (color >> 16) & 0xff;
                int g = (color >> 8) & 0xff;
                int b = color & 0xff;
                double brightness = (r + g + b) / 3.0;
                energy[y][x] = brightness;
            }
        }
        return energy;
    }

    private static int[] findSeamDp(double[][] energy) {
        int height = energy.length;
        int width = energy[0].length;

        double[][] dp = new double[height][width];
        int[][] path = new int[height][width];

        // Initialize the first row
        for (int x = 0; x < width; x++) {
            dp[0][x] = energy[0][x];
        }

        // Build the dp table
        for (int y = 1; y < height; y++) {
            for (int x = 0; x < width; x++) {
                double minEnergy = dp[y - 1][x];
                int minIndex = x;

                if (x > 0 && dp[y - 1][x - 1] < minEnergy) {
                    minEnergy = dp[y - 1][x - 1];
                    minIndex = x - 1;
                }
                if (x < width - 1 && dp[y - 1][x + 1] < minEnergy) {
                    minEnergy = dp[y - 1][x + 1];
                    minIndex = x + 1;
                }
                dp[y][x] = energy[y][x] + minEnergy;
                path[y][x] = minIndex;
            }
        }

        // Trace back to find the minimum energy seam
        int minIndex = 0;
        for (int x = 1; x < width; x++) {
            if (dp[height - 1][x] < dp[height - 1][minIndex]) {
                minIndex = x;
            }
        }
```

```
        int[] seam = new int[height];
        seam[height - 1] = minIndex;
        for (int y = height - 2; y >= 0; y--) {
            seam[y] = path[y + 1][seam[y + 1]];
        }
        return seam;
    }

    private static BufferedImage removeSeam(BufferedImage image, int[] seam) {
        int width = image.getWidth();
        int height = image.getHeight();
        BufferedImage newImage = new BufferedImage(width - 1, height, BufferedImage.TYPE_INT_RGB);

        for (int y = 0; y < height; y++) {
            int seamX = seam[y];
            for (int x = 0; x < width - 1; x++) {
                int srcX = (x < seamX) ? x : x + 1;
                newImage.setRGB(x, y, image.getRGB(srcX, y));
            }
        }
        return newImage;
    }

    private static void showSideBySide(BufferedImage before, BufferedImage after) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("Seam Carving - DP Result");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setLayout(new GridLayout(1, 2));

            JLabel beforeLabel = new JLabel(new ImageIcon(before));
            JLabel afterLabel = new JLabel(new ImageIcon(after));

            JPanel beforePanel = new JPanel(new BorderLayout());
            beforePanel.add(new JLabel("Original", SwingConstants.CENTER), BorderLayout.NORTH);
            beforePanel.add(beforeLabel, BorderLayout.CENTER);

            JPanel afterPanel = new JPanel(new BorderLayout());
            afterPanel.add(new JLabel("After Seam Carving", SwingConstants.CENTER), BorderLayout.NORTH);
            afterPanel.add(afterLabel, BorderLayout.CENTER);

            frame.add(beforePanel);
            frame.add(afterPanel);
            frame.pack();
            frame.setLocationRelativeTo(null);
            frame.setVisible(true);
        });
    }
}
```

CHALLENGES FACED AND HOW THEY WERE TACKLED

01

Challenge: It was difficult to correctly find and store the minimum-energy seam, because each pixel's optimal path depended on previous calculations.
Solution: I created an additional matrix to store the best previous pixel for each position. After calculating the cumulative energy, I used this matrix to trace back and reconstruct the full seam efficiently without confusion.

02

Challenge: Since the greedy method chooses the locally best option without considering the future, sometimes it removed seams passing through important parts of the image, distorting the main object.
Solution: I carefully adjusted the number of seams removed, and tested with different starting points to minimize damage. I also compared energy maps before removing seams to make better local choices.

03

Challenge: Computing the energy map using the Sobel operator was tricky. Small mistakes in handling image borders caused wrong energy values and affected seam selection.
Solution: I corrected the Sobel implementation by carefully handling image edge conditions (border pixels) and tested by visualizing the energy maps before seam removal to ensure accuracy.

TEAM WORK

Criteria	Sarah Alshddi	Nada Almalq	Sultanah Alahmed
Work division: Contributed equally to the work.	2	2	2
Peer evaluation: Level of commitments (Interactivity with other team members), and professional behavior towards team & TA	2	2	2
Project Discussion: Accurate answers, understanding of the presented work, good listeners to questions.	2	2	2
Time management: Attending on time, being ready to start the demo, good time management in discussion and demo.	2	2	2