DAA Lab Problems

# Dynamic Programming

### Longest common subsequence :

Code :

import java.util.Scanner;

public class LongestCommonSubsequence {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Take user input for two strings

System.out.println("Enter the first sequence:");

String X = sc.nextLine();

System.out.println("Enter the second sequence:");

String Y = sc.nextLine();

// Compute LCS

String lcs = findLCS(X, Y);

// Display results

System.out.println("Length of LCS: " + lcs.length());

System.out.println("LCS: " + lcs);

sc.close();

}

public static String findLCS(String X, String Y) {

int m = X.length();

int n = Y.length();

// Create a DP table

int[][] dp = new int[m + 1][n + 1];

// Fill the DP table

for (int i = 1; i <= m; i++) {

for (int j = 1; j <= n; j++) {

if (X.charAt(i - 1) == Y.charAt(j - 1)) {

dp[i][j] = dp[i - 1][j - 1] + 1;

} else {

dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);

}

}

}

// Backtrack to find the LCS string

StringBuilder lcs = new StringBuilder();

int i = m, j = n;

while (i > 0 && j > 0) {

if (X.charAt(i - 1) == Y.charAt(j - 1)) {

lcs.append(X.charAt(i - 1));

i--;

j--;

} else if (dp[i - 1][j] > dp[i][j - 1]) {

i--;

} else {

j--;

}

}

// Reverse the LCS string to get the correct order

return lcs.reverse().toString();

}

}

### Longest Common Substring :

Code :

import java.util.Scanner;

public class LongestCommonSubstring {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Take user input for two strings

System.out.println("Enter the first string:");

String X = sc.nextLine();

System.out.println("Enter the second string:");

String Y = sc.nextLine();

// Compute Longest Common Substring

String lcs = findLongestCommonSubstring(X, Y);

// Display results

System.out.println("Length of Longest Common Substring: " + lcs.length());

System.out.println("Longest Common Substring: " + lcs);

sc.close();

}

public static String findLongestCommonSubstring(String X, String Y) {

int m = X.length();

int n = Y.length();

// Create a DP table

int[][] dp = new int[m + 1][n + 1];

int maxLength = 0; // To track the maximum length

int endIndex = 0; // To track the ending index of the substring in X

// Fill the DP table

for (int i = 1; i <= m; i++) {

for (int j = 1; j <= n; j++) {

if (X.charAt(i - 1) == Y.charAt(j - 1)) {

dp[i][j] = dp[i - 1][j - 1] + 1;

// Update maxLength and endIndex if we find a longer substring

if (dp[i][j] > maxLength) {

maxLength = dp[i][j];

endIndex = i;

}

} else {

dp[i][j] = 0; // Reset to 0 if characters do not match

}

}

}

// Extract the longest common substring

return X.substring(endIndex - maxLength, endIndex);

}

}

### Matrix Multiplication :

Code :

import java.util.Scanner;

public class MatrixChainMultiplication {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Input the number of matrices

System.out.println("Enter the number of matrices:");

int n = sc.nextInt();

// Input the dimensions of matrices

int[] dimensions = new int[n + 1];

System.out.println("Enter the dimensions of the matrices:");

for (int i = 0; i <= n; i++) {

dimensions[i] = sc.nextInt();

}

// Compute the minimum number of multiplications

int minMultiplications = matrixChainOrder(dimensions);

System.out.println("Minimum number of multiplications: " + minMultiplications);

sc.close();

}

public static int matrixChainOrder(int[] dimensions) {

int n = dimensions.length - 1; // Number of matrices

int[][] dp = new int[n + 1][n + 1];

// Fill the DP table

for (int length = 2; length <= n; length++) { // length is the chain length

for (int i = 1; i <= n - length + 1; i++) {

int j = i + length - 1;

dp[i][j] = Integer.MAX\_VALUE;

// Try all possible places to split

for (int k = i; k < j; k++) {

int cost = dp[i][k] + dp[k + 1][j] + dimensions[i - 1] \* dimensions[k] \* dimensions[j];

dp[i][j] = Math.min(dp[i][j], cost);

}

}

}

return dp[1][n]; // Minimum cost to multiply matrices A1...An

}

}

### 0-1 knapsack problem :

Code :

import java.util.Scanner;

public class Knapsack01 {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Input number of items

System.out.println("Enter the number of items:");

int n = sc.nextInt();

// Input weights and values

int[] weights = new int[n];

int[] values = new int[n];

System.out.println("Enter the weights of the items:");

for (int i = 0; i < n; i++) {

weights[i] = sc.nextInt();

}

System.out.println("Enter the values of the items:");

for (int i = 0; i < n; i++) {

values[i] = sc.nextInt();

}

// Input maximum capacity of the knapsack

System.out.println("Enter the maximum capacity of the knapsack:");

int W = sc.nextInt();

// Compute the maximum value using DP

int maxValue = knapsack(weights, values, W);

System.out.println("The maximum value that can be obtained is: " + maxValue);

sc.close();

}

public static int knapsack(int[] weights, int[] values, int W) {

int n = weights.length;

int[][] dp = new int[n + 1][W + 1];

// Fill the DP table

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= W; j++) {

if (weights[i - 1] <= j) {

// Include or exclude the current item

dp[i][j] = Math.max(dp[i - 1][j], dp[i - 1][j - weights[i - 1]] + values[i - 1]);

} else {

// Exclude the current item

dp[i][j] = dp[i - 1][j];

}

}

}

return dp[n][W];

}

}

### Subset Sum Problem – Dynamic Programming Solution :

Code :

import java.util.ArrayList;

import java.util.Scanner;

public class SubsetSum {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Input the number of elements in the set

System.out.println("Enter the number of elements in the set:");

int n = sc.nextInt();

// Input the elements of the set

int[] arr = new int[n];

System.out.println("Enter the elements of the set:");

for (int i = 0; i < n; i++) {

arr[i] = sc.nextInt();

}

// Input the target sum

System.out.println("Enter the target sum:");

int k = sc.nextInt();

// Check if there is a subset with the given sum

ArrayList<Integer> subset = findSubset(arr, n, k);

if (subset != null) {

System.out.println("Yes, there exists a subset with the given sum.");

System.out.println("Subset: " + subset);

} else {

System.out.println("No, there does not exist a subset with the given sum.");

}

sc.close();

}

public static ArrayList<Integer> findSubset(int[] arr, int n, int k) {

// DP table to store results

boolean[][] dp = new boolean[n + 1][k + 1];

// Base case: sum 0 can be achieved with an empty subset

for (int i = 0; i <= n; i++) {

dp[i][0] = true;

}

// Fill the DP table

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= k; j++) {

if (arr[i - 1] <= j) {

dp[i][j] = dp[i - 1][j] || dp[i - 1][j - arr[i - 1]];

} else {

dp[i][j] = dp[i - 1][j];

}

}

}

// If no subset is found

if (!dp[n][k]) {

return null;

}

// Backtrack to find the subset

ArrayList<Integer> subset = new ArrayList<>();

int i = n, j = k;

while (i > 0 && j > 0) {

// Check if the current item was included

if (dp[i][j] && !dp[i - 1][j]) {

subset.add(arr[i - 1]);

j -= arr[i - 1];

}

i--;

}

return subset;

}

}

### Rod Cutting Problem :

Code :

import java.util.Scanner;

public class RodCutting {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Input the length of the rod

System.out.println("Enter the length of the rod:");

int n = sc.nextInt();

// Input the prices for each length

int[] prices = new int[n];

System.out.println("Enter the prices for each length:");

for (int i = 0; i < n; i++) {

prices[i] = sc.nextInt();

}

// Calculate the maximum profit

int maxProfit = rodCutting(prices, n);

System.out.println("The maximum profit is: " + maxProfit);

sc.close();

}

public static int rodCutting(int[] prices, int n) {

// DP array to store the maximum profit for each rod length

int[] dp = new int[n + 1];

// Base case: profit for a rod of length 0 is 0

dp[0] = 0;

// Fill the DP array

for (int i = 1; i <= n; i++) {

int maxVal = Integer.MIN\_VALUE;

for (int j = 1; j <= i; j++) {

maxVal = Math.max(maxVal, prices[j - 1] + dp[i - j]);

}

dp[i] = maxVal;

}

return dp[n];

}

}

# Divide and Conquer :

### Inversion Count :

Code :

import java.util.Scanner;

public class CountInversions {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

// Input array size

System.out.println("Enter the size of the array:");

int n = sc.nextInt();

// Input array elements

int[] arr = new int[n];

System.out.println("Enter the elements of the array:");

for (int i = 0; i < n; i++) {

arr[i] = sc.nextInt();

}

// Count inversions

int[] temp = new int[n];

int inversions = mergeSortAndCount(arr, temp, 0, n - 1);

System.out.println("Total number of inversions: " + inversions);

sc.close();

}

// Function to count inversions using merge sort

public static int mergeSortAndCount(int[] arr, int[] temp, int left, int right) {

int invCount = 0;

if (left < right) {

int mid = (left + right) / 2;

// Count inversions in left half

invCount += mergeSortAndCount(arr, temp, left, mid);

// Count inversions in right half

invCount += mergeSortAndCount(arr, temp, mid + 1, right);

// Count split inversions

invCount += mergeAndCount(arr, temp, left, mid, right);

}

return invCount;

}

// Merge function that counts split inversions

public static int mergeAndCount(int[] arr, int[] temp, int left, int mid, int right) {

int i = left; // Starting index for left subarray

int j = mid + 1; // Starting index for right subarray

int k = left; // Starting index to store sorted elements

int invCount = 0;

// Merge and count inversions

while (i <= mid && j <= right) {

if (arr[i] <= arr[j]) {

temp[k++] = arr[i++];

} else {

temp[k++] = arr[j++];

invCount += (mid - i + 1); // Count inversions

}

}

// Copy remaining elements of left subarray

while (i <= mid) {

temp[k++] = arr[i++];

}

// Copy remaining elements of right subarray

while (j <= right) {

temp[k++] = arr[j++];

}

// Copy sorted elements back to original array

for (i = left; i <= right; i++) {

arr[i] = temp[i];

}

return invCount;

}

}

### Find the first or last occurrence of a given number in a sorted array :

Code :

import java.util.Scanner;

public class FirstLastOccurrence {

// Function to find the first occurrence of a number

public static int findFirstOccurrence(int[] arr, int target) {

int left = 0, right = arr.length - 1;

int result = -1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == target) {

result = mid; // Update result and search left for first occurrence

right = mid - 1;

} else if (arr[mid] < target) {

left = mid + 1; // Search in the right half

} else {

right = mid - 1; // Search in the left half

}

}

return result;

}

// Function to find the last occurrence of a number

public static int findLastOccurrence(int[] arr, int target) {

int left = 0, right = arr.length - 1;

int result = -1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == target) {

result = mid; // Update result and search right for last occurrence

left = mid + 1;

} else if (arr[mid] < target) {

left = mid + 1; // Search in the right half

} else {

right = mid - 1; // Search in the left half

}

}

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input the size of the array

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

// Input the sorted array

int[] arr = new int[n];

System.out.println("Enter the sorted array elements:");

for (int i = 0; i < n; i++) {

arr[i] = scanner.nextInt();

}

// Ensure the buffer is clear

scanner.nextLine();

// Input the target number

System.out.print("Enter the target number: ");

int target = scanner.nextInt();

// Find first and last occurrences

int first = findFirstOccurrence(arr, target);

int last = findLastOccurrence(arr, target);

// Display the results

if (first != -1) {

System.out.println("First occurrence of " + target + ": " + first);

System.out.println("Last occurrence of " + target + ": " + last);

} else {

System.out.println("Number " + target + " is not present in the array.");

}

scanner.close();

}

}

### Find the smallest missing element from a sorted array :

Code :

import java.util.Scanner;

public class SmallestMissingElement {

// Function to find the smallest missing non-negative element

public static int findSmallestMissing(int[] arr) {

int left = 0, right = arr.length - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

// If the element matches its index, the missing element is in the right half

if (arr[mid] == mid) {

left = mid + 1;

} else {

// Otherwise, it is in the left half

right = mid - 1;

}

}

// The smallest missing element is at the `left` index

return left;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input the size of the array

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

// Input the sorted array of distinct non-negative integers

int[] arr = new int[n];

System.out.println("Enter the sorted array elements:");

for (int i = 0; i < n; i++) {

arr[i] = scanner.nextInt();

}

// Find and display the smallest missing non-negative element

int smallestMissing = findSmallestMissing(arr);

System.out.println("The smallest missing non-negative element is: " + smallestMissing);

scanner.close();

}

}

### Find the number of 1’s in a sorted binary array :

Code :

import java.util.Scanner;

public class CountOnesInBinaryArray {

// Function to find the index of the first occurrence of 1

public static int findFirstOne(int[] arr) {

int left = 0, right = arr.length - 1;

int result = -1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == 1) {

result = mid; // Update result and search in the left half

right = mid - 1;

} else {

left = mid + 1; // Search in the right half

}

}

return result; // Returns the index of the first 1 or -1 if no 1 is found

}

// Function to count the total number of 1's in the binary array

public static int countOnes(int[] arr) {

int firstOneIndex = findFirstOne(arr);

// If no 1's are found, return 0

if (firstOneIndex == -1) {

return 0;

}

// Total number of 1's is from the first 1 to the end of the array

return arr.length - firstOneIndex;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input the size of the array

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

// Input the sorted binary array

int[] arr = new int[n];

System.out.println("Enter the sorted binary array elements (0s and 1s only):");

for (int i = 0; i < n; i++) {

arr[i] = scanner.nextInt();

}

// Count and display the total number of 1's

int totalOnes = countOnes(arr);

System.out.println("Total number of 1's in the array: " + totalOnes);

scanner.close();

}

}

### Find pairs with difference `k` in an array | Constant Space Solution :

Code :

import java.util.Arrays;

import java.util.Scanner;

public class PairsWithGivenDifference {

// Function to find all pairs with a given difference k

public static void findPairsWithDifference(int[] arr, int k) {

// Sort the array (O(n log n))

Arrays.sort(arr);

int i = 0, j = 1;

System.out.println("Pairs with difference " + k + ":");

// Two-pointer approach

while (i < arr.length && j < arr.length) {

int diff = arr[j] - arr[i];

if (diff == k && i != j) {

// Pair found

System.out.println("(" + arr[i] + ", " + arr[j] + ")");

i++;

j++;

} else if (diff < k) {

// Increase the larger pointer to increase the difference

j++;

} else {

// Increase the smaller pointer to decrease the difference

i++;

}

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input the size of the array

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

// Input the array

int[] arr = new int[n];

System.out.println("Enter the array elements:");

for (int i = 0; i < n; i++) {

arr[i] = scanner.nextInt();

}

// Input the difference value

System.out.print("Enter the difference value (k): ");

int k = scanner.nextInt();

// Find and display pairs

findPairsWithDifference(arr, k);

scanner.close();

}

}

### Find `k` closest elements to a given value in an array :

Code :

import java.util.Scanner;

import java.util.ArrayList;

import java.util.List;

public class KClosestElements {

// Function to find the index of the closest element using binary search

public static int findClosestIndex(int[] arr, int target) {

int left = 0, right = arr.length - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == target) {

return mid;

} else if (arr[mid] < target) {

left = mid + 1;

} else {

right = mid - 1;

}

}

// Return the index of the closest element

if (left >= arr.length) {

return arr.length - 1;

}

if (right < 0) {

return 0;

}

return (Math.abs(arr[left] - target) < Math.abs(arr[right] - target)) ? left : right;

}

// Function to find the k closest elements to the target

public static List<Integer> findKClosest(int[] arr, int target, int k) {

List<Integer> result = new ArrayList<>();

int closestIndex = findClosestIndex(arr, target);

int left = closestIndex;

int right = closestIndex + 1;

// Collect k closest elements

while (k > 0) {

if (left >= 0 && right < arr.length) {

if (Math.abs(arr[left] - target) <= Math.abs(arr[right] - target)) {

result.add(arr[left]);

left--;

} else {

result.add(arr[right]);

right++;

}

} else if (left >= 0) {

result.add(arr[left]);

left--;

} else if (right < arr.length) {

result.add(arr[right]);

right++;

}

k--;

}

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input the size of the array

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

// Input the sorted array elements

int[] arr = new int[n];

System.out.println("Enter the sorted array elements:");

for (int i = 0; i < n; i++) {

arr[i] = scanner.nextInt();

}

// Input the target and k values

System.out.print("Enter the target value: ");

int target = scanner.nextInt();

System.out.print("Enter the value of k: ");

int k = scanner.nextInt();

// Find and display the k closest elements to the target

List<Integer> result = findKClosest(arr, target, k);

System.out.println("The " + k + " closest elements to " + target + " are:");

for (int num : result) {

System.out.print(num + " ");

}

scanner.close();

}

}

### 7)Longest Common Prefix (LCP) Problem :

Code :

import java.util.Scanner;

public class LongestCommonPrefix {

// Function to find the LCP of two strings

public static String commonPrefix(String str1, String str2) {

int minLength = Math.min(str1.length(), str2.length());

int i = 0;

while (i < minLength && str1.charAt(i) == str2.charAt(i)) {

i++;

}

return str1.substring(0, i); // Return the common prefix

}

// Function to find the LCP of an array of strings using divide and conquer

public static String longestCommonPrefix(String[] strs, int left, int right) {

if (left == right) {

return strs[left]; // Base case: only one string

} else {

int mid = left + (right - left) / 2;

// Divide the problem into two halves

String leftLCP = longestCommonPrefix(strs, left, mid);

String rightLCP = longestCommonPrefix(strs, mid + 1, right);

// Conquer: find the common prefix of the two halves

return commonPrefix(leftLCP, rightLCP);

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Take user input for the number of strings

System.out.print("Enter the number of strings: ");

int n = scanner.nextInt();

scanner.nextLine(); // Consume newline left by nextInt()

// Take user input for the strings

String[] strs = new String[n];

System.out.println("Enter the strings:");

for (int i = 0; i < n; i++) {

strs[i] = scanner.nextLine();

}

// Find and print the longest common prefix

if (strs.length == 0) {

System.out.println("No common prefix");

} else {

String result = longestCommonPrefix(strs, 0, strs.length - 1);

System.out.println("Longest Common Prefix: " + result);

}

scanner.close(); // Close the scanner

}

}

### 8)Find the frequency of each element in a sorted array containing duplicates:

Code :

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

public class FrequencyFinder {

// Helper function to perform binary search for the first occurrence of an element

private static int findFirstOccurrence(int[] arr, int target) {

int low = 0, high = arr.length - 1;

int result = -1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == target) {

result = mid;

high = mid - 1; // Continue searching on the left side

} else if (arr[mid] < target) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return result;

}

// Helper function to perform binary search for the last occurrence of an element

private static int findLastOccurrence(int[] arr, int target) {

int low = 0, high = arr.length - 1;

int result = -1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == target) {

result = mid;

low = mid + 1; // Continue searching on the right side

} else if (arr[mid] < target) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return result;

}

// Function to find the frequency of each element in the sorted array

public static Map<Integer, Integer> findFrequencies(int[] arr) {

Map<Integer, Integer> frequencyMap = new HashMap<>();

int n = arr.length;

int i = 0;

// Traverse the array using divide and conquer (binary search for each element)

while (i < n) {

int target = arr[i];

// Find the first and last occurrence of the target element

int firstOccurrence = findFirstOccurrence(arr, target);

int lastOccurrence = findLastOccurrence(arr, target);

// Calculate the frequency and store it in the map

int frequency = lastOccurrence - firstOccurrence + 1;

frequencyMap.put(target, frequency);

// Skip the elements that have been counted

i = lastOccurrence + 1;

}

return frequencyMap;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Take user input for the size of the array

System.out.print("Enter the size of the sorted array: ");

int n = scanner.nextInt();

// Initialize the array

int[] arr = new int[n];

// Take user input for the elements of the sorted array

System.out.println("Enter the elements of the sorted array:");

for (int i = 0; i < n; i++) {

arr[i] = scanner.nextInt();

}

// Find the frequencies

Map<Integer, Integer> frequencies = findFrequencies(arr);

// Output the frequencies of each element

System.out.println("Element frequencies:");

for (Map.Entry<Integer, Integer> entry : frequencies.entrySet()) {

System.out.println("Element: " + entry.getKey() + ", Frequency: " + entry.getValue());

}

scanner.close(); // Close the scanner

} }

### Maximum Subarray Sum using Divide and Conquer :

Code :

import java.util.Scanner;

public class MaxSubarraySum {

// Function to find the maximum sum of the subarray that crosses the midpoint

private static int findMaxCrossingSum(int[] arr, int low, int mid, int high) {

// Left half (starting from mid and going left)

int leftSum = Integer.MIN\_VALUE;

int sum = 0;

for (int i = mid; i >= low; i--) {

sum += arr[i];

if (sum > leftSum) {

leftSum = sum;

}

}

// Right half (starting from mid+1 and going right)

int rightSum = Integer.MIN\_VALUE;

sum = 0;

for (int i = mid + 1; i <= high; i++) {

sum += arr[i];

if (sum > rightSum) {

rightSum = sum;

}

}

// Return the sum of the left, right, and the middle crossing subarrays

return leftSum + rightSum;

}

// Divide and conquer function to find the maximum subarray sum

private static int maxSubArraySum(int[] arr, int low, int high) {

if (low == high) {

// Base case: only one element

return arr[low];

} else {

// Find the midpoint

int mid = (low + high) / 2;

// Find the maximum sum in the left half, right half, and crossing subarray

int leftSum = maxSubArraySum(arr, low, mid);

int rightSum = maxSubArraySum(arr, mid + 1, high);

int crossingSum = findMaxCrossingSum(arr, low, mid, high);

// Return the maximum of the three

return Math.max(Math.max(leftSum, rightSum), crossingSum);

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Take user input for the size of the array

System.out.print("Enter the size of the array: ");

int n = scanner.nextInt();

// Initialize the array

int[] arr = new int[n];

// Take user input for the elements of the array

System.out.println("Enter the elements of the array:");

for (int i = 0; i < n; i++) {

arr[i] = scanner.nextInt();

}

// Find the maximum subarray sum

int maxSum = maxSubArraySum(arr, 0, arr.length - 1);

// Output the result

System.out.println("Maximum Subarray Sum: " + maxSum);

scanner.close(); // Close the scanner

}

}

# Greedy Algorithm :

### Graph Coloring Problem :

Code :

import java.util.\*;

public class GraphColoring {

// Function to perform greedy coloring

public static void greedyColoring(int[][] graph, int numVertices) {

// Result array to store the color of each vertex

int[] result = new int[numVertices];

// Initially, no vertex is colored

Arrays.fill(result, -1);

// Assign the first color to the first vertex

result[0] = 0;

// Temporary array to store the available colors

boolean[] available = new boolean[numVertices];

// Assign a color to each vertex

for (int u = 1; u < numVertices; u++) {

// Reset the available colors

Arrays.fill(available, true);

// Mark the colors of the adjacent vertices as unavailable

for (int v = 0; v < numVertices; v++) {

if (graph[u][v] == 1 && result[v] != -1) {

available[result[v]] = false;

}

}

// Find the first available color

int color;

for (color = 0; color < numVertices; color++) {

if (available[color]) {

break;

}

}

// Assign the found color to the current vertex

result[u] = color;

}

// Print the result

System.out.println("Vertex\tColor");

for (int u = 0; u < numVertices; u++) {

System.out.println(u + "\t" + result[u]);

}

}

// Function to take user input for the graph and the number of vertices

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Take user input for the number of vertices

System.out.print("Enter the number of vertices: ");

int numVertices = scanner.nextInt();

// Initialize the graph as an adjacency matrix

int[][] graph = new int[numVertices][numVertices];

// Take user input for the edges of the graph

System.out.println("Enter the adjacency matrix (0 for no edge, 1 for edge):");

for (int i = 0; i < numVertices; i++) {

for (int j = 0; j < numVertices; j++) {

graph[i][j] = scanner.nextInt();

}

}

// Call the greedy coloring function

greedyColoring(graph, numVertices);

scanner.close(); // Close the scanner

}

}

### Huffman Coding Compression Algorithm :

Code :

import java.util.\*;

// Huffman Tree Node class

class Node {

char character;

int frequency;

Node left, right;

// Constructor

public Node(char character, int frequency) {

this.character = character;

this.frequency = frequency;

left = right = null;

}

}

// Comparator class to help with priority queue sorting

class NodeComparator implements Comparator<Node> {

public int compare(Node node1, Node node2) {

return node1.frequency - node2.frequency;

}

}

public class HuffmanCoding {

// Function to build the Huffman Tree

public static Node buildHuffmanTree(Map<Character, Integer> freqMap) {

PriorityQueue<Node> minHeap = new PriorityQueue<>(new NodeComparator());

// Create a leaf node for each character and add it to the min-heap

for (Map.Entry<Character, Integer> entry : freqMap.entrySet()) {

minHeap.offer(new Node(entry.getKey(), entry.getValue()));

}

// Build the Huffman Tree

while (minHeap.size() > 1) {

// Extract two nodes with the lowest frequency

Node left = minHeap.poll();

Node right = minHeap.poll();

// Create a new internal node with a frequency equal to the sum of the two nodes' frequencies

Node internalNode = new Node('-', left.frequency + right.frequency);

internalNode.left = left;

internalNode.right = right;

// Insert the internal node back into the priority queue

minHeap.offer(internalNode);

}

// The remaining node is the root of the Huffman Tree

return minHeap.poll();

}

// Function to generate the Huffman codes

public static void generateHuffmanCodes(Node root, StringBuilder prefix, Map<Character, String> huffmanCodes) {

// Base case: If we reach a leaf node, store the code for the character

if (root == null) {

return;

}

// If the node is a leaf, assign the current prefix as the Huffman code for the character

if (root.left == null && root.right == null) {

huffmanCodes.put(root.character, prefix.toString());

}

// Recur for left and right subtrees

generateHuffmanCodes(root.left, prefix.append('0'), huffmanCodes);

prefix.deleteCharAt(prefix.length() - 1); // Remove last character after recursion

generateHuffmanCodes(root.right, prefix.append('1'), huffmanCodes);

prefix.deleteCharAt(prefix.length() - 1); // Remove last character after recursion

}

// Function to perform Huffman Coding

public static void huffmanCoding(String input) {

// Step 1: Calculate the frequency of each character in the input string

Map<Character, Integer> freqMap = new HashMap<>();

for (char c : input.toCharArray()) {

freqMap.put(c, freqMap.getOrDefault(c, 0) + 1);

}

// Step 2: Build the Huffman Tree

Node root = buildHuffmanTree(freqMap);

// Step 3: Generate Huffman codes for each character

Map<Character, String> huffmanCodes = new HashMap<>();

generateHuffmanCodes(root, new StringBuilder(), huffmanCodes);

// Step 4: Display the results

System.out.println("Character | Frequency | Huffman Code");

for (Map.Entry<Character, String> entry : huffmanCodes.entrySet()) {

System.out.println(entry.getKey() + " | " + freqMap.get(entry.getKey()) + " | " + entry.getValue());

}

// Step 5: Display the encoded output

StringBuilder encodedOutput = new StringBuilder();

for (char c : input.toCharArray()) {

encodedOutput.append(huffmanCodes.get(c));

}

System.out.println("\nEncoded output: " + encodedOutput);

System.out.println("Original input size: " + input.length() \* 8 + " bits");

System.out.println("Encoded output size: " + encodedOutput.length() + " bits");

System.out.println("Compression ratio: " + (1.0 - (encodedOutput.length() / (input.length() \* 8.0))) \* 100 + "%");

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Take user input for the string

System.out.print("Enter the string to compress: ");

String input = scanner.nextLine();

// Perform Huffman coding

huffmanCoding(input);

scanner.close();

}

}

### Kruskal’s and Prim’s Algorithm for finding Minimum Spanning Tree :

Code :

import java.util.\*;

class Kruskal {

// Disjoint Set Union-Find structure to detect cycles

static class UnionFind {

int[] parent, rank;

// Initialize the Union-Find structure

public UnionFind(int n) {

parent = new int[n];

rank = new int[n];

for (int i = 0; i < n; i++) {

parent[i] = i;

rank[i] = 0;

}

}

// Find the root of the set containing 'x'

public int find(int x) {

if (parent[x] != x) {

parent[x] = find(parent[x]); // Path compression

}

return parent[x];

}

// Union of two sets

public void union(int x, int y) {

int rootX = find(x);

int rootY = find(y);

// Union by rank

if (rootX != rootY) {

if (rank[rootX] > rank[rootY]) {

parent[rootY] = rootX;

} else if (rank[rootX] < rank[rootY]) {

parent[rootX] = rootY;

} else {

parent[rootY] = rootX;

rank[rootX]++;

}

}

}

}

// Kruskal's Algorithm to find MST

public static void kruskal(int V, List<int[]> edges) {

UnionFind uf = new UnionFind(V);

edges.sort(Comparator.comparingInt(e -> e[2])); // Sort by edge weight

int mstWeight = 0;

List<int[]> mstEdges = new ArrayList<>();

// Iterate through the edges, adding the smallest edge if no cycle is formed

for (int[] edge : edges) {

int u = edge[0];

int v = edge[1];

int weight = edge[2];

if (uf.find(u) != uf.find(v)) {

uf.union(u, v);

mstEdges.add(edge);

mstWeight += weight;

}

}

// Print MST edges and total weight

System.out.println("Kruskal's MST edges (u, v, weight):");

for (int[] edge : mstEdges) {

System.out.println("(" + edge[0] + ", " + edge[1] + ", " + edge[2] + ")");

}

System.out.println("Total weight of MST: " + mstWeight);

}

}

class Prim {

// Prim's Algorithm to find MST

public static void prim(int V, int[][] graph) {

boolean[] inMST = new boolean[V];

int[] key = new int[V];

int[] parent = new int[V];

Arrays.fill(key, Integer.MAX\_VALUE);

key[0] = 0; // Start from vertex 0

parent[0] = -1; // Root of MST doesn't have a parent

PriorityQueue<int[]> pq = new PriorityQueue<>(Comparator.comparingInt(a -> a[1]));

pq.offer(new int[]{0, 0}); // {vertex, key}

int mstWeight = 0;

while (!pq.isEmpty()) {

int u = pq.poll()[0];

inMST[u] = true;

// Update adjacent vertices

for (int v = 0; v < V; v++) {

if (graph[u][v] != 0 && !inMST[v] && graph[u][v] < key[v]) {

key[v] = graph[u][v];

pq.offer(new int[]{v, key[v]});

parent[v] = u;

}

}

}

// Print MST edges and total weight

System.out.println("Prim's MST edges (u, v, weight):");

for (int i = 1; i < V; i++) {

System.out.println("(" + parent[i] + ", " + i + ", " + graph[i][parent[i]] + ")");

mstWeight += graph[i][parent[i]];

}

System.out.println("Total weight of MST: " + mstWeight);

}

}

public class MST {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Take user input for the number of vertices

System.out.print("Enter the number of vertices: ");

int V = scanner.nextInt();

// Take user input for the number of edges

System.out.print("Enter the number of edges: ");

int E = scanner.nextInt();

List<int[]> edges = new ArrayList<>();

int[][] graph = new int[V][V];

System.out.println("Enter the edges (u, v, weight): ");

for (int i = 0; i < E; i++) {

int u = scanner.nextInt();

int v = scanner.nextInt();

int weight = scanner.nextInt();

edges.add(new int[]{u, v, weight});

graph[u][v] = weight;

graph[v][u] = weight; // Since it's an undirected graph

}

// Run Kruskal's Algorithm

System.out.println("\nRunning Kruskal's Algorithm:");

Kruskal.kruskal(V, edges);

// Run Prim's Algorithm

System.out.println("\nRunning Prim's Algorithm:");

Prim.prim(V, graph);

scanner.close(); // Close the scanner

}

}