**TASK 1 – MATHEMATICS – WHY MATHS IN CODING?**

**1.A**

**PROGRAM:**

import java.util.\*;

class task1a {

public static void main(String args[]) {

Scanner scan = new Scanner(System.in);

int n = scan.nextInt();

double value = Math.sqrt(n);

int s = (int) value;

System.out.println(s);

}

}

**1.B**

**PROGRAM:**

import java.util.\*;

class task1b {

public static void main(String args[]) {

Scanner scan = new Scanner(System.in);

int num = scan.nextInt();

int temp = num;

while (temp % 2 == 0) {

temp /= 2;

}

while (temp % 3 == 0) {

temp /= 3;

}

while (temp % 5 == 0) {

temp /= 5;

}

if (temp == 1) {

System.out.println(num + " is an ugly number");

} else {

System.out.println(num + " is not an ugly number");

}

}

}

**1.C**

**PROGRAM:**

import java.util.Scanner;

class task1c {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

int[] arr = new int[n];

System.out.println("Enter elements");

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

arr[i] = sc.nextInt();

}

int result = 0;

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

result += arr[i];

}

System.out.println("Sum of all elements: " + result);

}

}

**TASK NO: 2 :- ADVANCED CONCEPTS ON ARRAY**

**2.A**

**PROGRAM:**

import java.util.\*;

public class IntervalIntersections {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int m = sc.nextInt();

int[][] first = new int[m][2];

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

first[i][0] = sc.nextInt();

first[i][1] = sc.nextInt();

}

int n = sc.nextInt();

int[][] second = new int[n][2];

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

second[i][0] = sc.nextInt();

second[i][1] = sc.nextInt();

}

int i = 0, j = 0;

while (i < m && j < n) {

int start = Math.max(first[i][0], second[j][0]);

int end = Math.min(first[i][1], second[j][1]);

if (start <= end) {

System.out.print("[" + start + ", " + end + "] ");

}

if (first[i][1] < second[j][1]) {

i++;

} else {

j++;

}

}

sc.close();

}

}

**2.B**

**PROGRAM:**

import java.util.\*;

public class MergeSortedArray {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int m = sc.nextInt();

int n = sc.nextInt();

int[] a = new int[m + n];

int[] b = new int[n];

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

a[i] = sc.nextInt();

}

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

b[i] = sc.nextInt();

}

int i = m - 1;

int j = n - 1;

int k = m + n - 1;

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

if (a[i] > b[j]) {

a[k--] = a[i--];

} else {

a[k--] = b[j--];

}

}

while (j >= 0) {

a[k--] = b[j--];

}

for (int x = 0; x < a.length; x++) {

System.out.print(a[x] + " ");

}

sc.close();

}

}

**2.C**

**PROGRAM:**

import java.util.\*;

public class TripletSum {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] arr = new int[n];

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

arr[i] = sc.nextInt();

}

Arrays.sort(arr);

for (int i = 0; i < n - 2; i++) {

if (i > 0 && arr[i] == arr[i - 1]) {

continue;

}

int l = i + 1;

int r = n - 1;

int target = -arr[i];

while (l < r) {

int sum = arr[l] + arr[r];

if (sum == target) {

System.out.println(arr[i] + " " + arr[l] + " " + arr[r]);

while (l < r && arr[l] == arr[l + 1]) {

l++;

}

while (l < r && arr[r] == arr[r - 1]) {

r--;

}

l++;

r--;

} else if (sum < target) {

l++;

} else {

r--;

}

}

}

sc.close();

}

}

**TASK 3 : ADVANCED CONCEPTS ON STRINGS**

**3.A**

**PROGRAM:**

import java.util.\*;

public class PatternFinding {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

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

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

System.out.print("\* ");

}

System.out.println();

}

sc.close();

}

}

**3.B**

**PROGRAM:**

import java.util.\*;

public class PalindromeCheck {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String input = sc.nextLine();

input = input.toLowerCase();

String reversed = new StringBuilder(input).reverse().toString();

if (input.equals(reversed)) {

System.out.println(input + " is a Palindrome.");

} else {

System.out.println(input + " is not a Palindrome.");

}

sc.close();

}

}

**3.C**

**PROGRAM:**

import java.util.\*;

public class PasswordValidation {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter a password to validate:");

String password = sc.nextLine();

String message = "";

boolean isValid = true;

if (password.length() < 8) {

message += "Password must be at least 8 characters long.\n";

isValid = false;

}

if (!password.matches(".[A-Z].")) {

message += "Password must contain at least one uppercase letter.\n";

isValid = false;

}

if (!password.matches(".[a-z].")) {

message += "Password must contain at least one lowercase letter.\n";

isValid = false;

}

if (!password.matches(".\\d.")) {

message += "Password must contain at least one digit.\n";

isValid = false;

}

if (!password.matches(".[@#$%^&+=].")) {

message += "Password must contain at least one special character from @#$%^&+=.\n";

isValid = false;

}

if (isValid) {

System.out.println("Password is valid.");

} else {

System.out.println("Invalid Password. The following criteria were not met:\n" + message);

}

sc.close();

}

}

**TASK 4 : STACK AND QUEUES**

**4.A**

**PROGRAM:**

import java.util.\*;

public class StackTwoQueues {

Queue<Integer> q1;

Queue<Integer> q2;

public StackTwoQueues() {

q1 = new LinkedList<>();

q2 = new LinkedList<>();

}

public void push(int x) {

q2.add(x);

while (!q1.isEmpty()) {

q2.add(q1.remove());

}

Queue<Integer> temp = q1;

q1 = q2;

q2 = temp;

}

public int pop() {

if (q1.isEmpty()) {

System.out.println("Stack is empty");

return -1;

}

return q1.remove();

}

public int peek() {

if (q1.isEmpty()) {

System.out.println("Stack is empty");

return -1;

}

return q1.peek();

}

public boolean isEmpty() {

return q1.isEmpty();

}

public static void main(String[] args) {

StackTwoQueues s = new StackTwoQueues();

s.push(10);

s.push(20);

s.push(30);

System.out.println(s.pop());

System.out.println(s.pop());

System.out.println(s.pop());

}

}

**4.B**

**PROGRAM:**

import java.util.\*;

class Bag {

List<Integer> bag;

public Bag() {

bag = new ArrayList<>();

}

public void add(int x) {

bag.add(x);

}

public void remove(int x) {

bag.remove(Integer.valueOf(x));

}

public int count(int x) {

int c = 0;

for (int num : bag) {

if (num == x) {

c++;

}

}

return c;

}

public boolean isEmpty() {

return bag.isEmpty();

}

public int size() {

return bag.size();

}

public void show() {

System.out.println(bag);

}

}

public class Main {

public static void main(String[] args) {

Bag b = new Bag();

b.add(5);

b.add(3);

b.add(5);

b.add(2);

b.show();

System.out.println("Size: " + b.size());

System.out.println("Count of 5: " + b.count(5));

b.remove(5);

b.show();

System.out.println("Empty: " + b.isEmpty());

}

}

**4.C**

**PROGRAM:**

import java.util.\*;

class DiskTower {

static void tower(int[] d, int day, int maxDisk, int currentCount, List<Integer> availableDisks) {

if (day > d.length) {

return;

}

System.out.println("Day " + (day + 1) + ":");

System.out.println("Plate " + (d[day]));

int nextDisk = d[day];

if (nextDisk <= maxDisk && availableDisks.contains(nextDisk)) {

availableDisks.remove(Integer.valueOf(nextDisk));

maxDisk = nextDisk;

currentCount++;

}

System.out.println("Current tower: " + availableDisks);

System.out.println("Count: " + currentCount);

tower(d, day + 1, maxDisk, currentCount, availableDisks);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] d = new int[n];

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

d[i] = sc.nextInt();

}

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

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

allDisks.add(i);

}

tower(d, 0, n + 1, 0, allDisks);

}

}

**Task 5: Linked List-Single linked list, Doubly liked list, operations on linked list.**

**5.a**

**PROGRAM:**

class InsertionSortLinkedList {

static class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

public static Node insertionSort(Node head) {

Node dummy = new Node(0);

Node current = head;

while (current != null) {

Node next = current.next;

Node prev = dummy;

while (prev.next != null && prev.next.data < current.data) {

prev = prev.next;

}

current.next = prev.next;

prev.next = current;

current = next;

}

return dummy.next;

}

public static void printList(Node head) {

Node temp = head;

while (temp != null) {

System.out.print(temp.data + " ");

temp = temp.next;

}

System.out.println();

}

public static void main(String[] args) {

Node head = new Node(4);

head.next = new Node(3);

head.next.next = new Node(1);

head.next.next.next = new Node(5);

head.next.next.next.next = new Node(2);

System.out.println("Original List:");

printList(head);

head = insertionSort(head);

System.out.println("Sorted List:");

printList(head);

}

}

**5.b**

**PROGRAM:**

import java.util.Scanner;

class RemoveElementLinkedList {

static class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

public static Node removeElement(Node head, int key) {

if (head == null) {

return null;

}

if (head.data == key) {

return head.next;

}

Node current = head;

Node prev = null;

while (current != null && current.data != key) {

prev = current;

current = current.next;

}

if (current != null) {

prev.next = current.next;

}

return head;

}

public static void printList(Node head) {

Node temp = head;

while (temp != null) {

System.out.print(temp.data + " ");

temp = temp.next;

}

System.out.println();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

if (n <= 0) {

System.out.println("List is empty.");

return;

}

System.out.println("Enter " + n + " elements:");

Node head = new Node(sc.nextInt());

Node temp = head;

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

temp.next = new Node(sc.nextInt());

temp = temp.next;

}

System.out.println("Original List:");

printList(head);

System.out.print("Enter element to remove: ");

int key = sc.nextInt();

head = removeElement(head, key);

System.out.println("Updated List:");

printList(head);

}

}

**5.c**

**PROGRAM:**

import java.util.\*;

class RemoveDuplicatesLinkedList {

static class Node {

int data;

Node next;

Node(int data) {

this.data = data;

this.next = null;

}

}

public static Node removeDuplicates(Node head) {

if (head == null) return null;

Set<Integer> set = new HashSet<>();

Node current = head;

Node prev = null;

while (current != null) {

if (set.contains(current.data)) {

prev.next = current.next;

} else {

set.add(current.data);

prev = current;

}

current = current.next;

}

return head;

}

public static void printList(Node head) {

Node temp = head;

while (temp != null) {

System.out.print(temp.data + " ");

temp = temp.next;

}

System.out.println();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

if (n <= 0) {

System.out.println("List is empty.");

return;

}

System.out.println("Enter " + n + " elements:");

Node head = new Node(sc.nextInt());

Node temp = head;

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

temp.next = new Node(sc.nextInt());

temp = temp.next;

}

System.out.println("Original List:");

printList(head);

head = removeDuplicates(head);

System.out.println("List after removing duplicates:");

printList(head);

}

}

**Task 6: Sorting- Counting sort, Radix sort, Heap sort, Bucket sort**

**6.a**

**PROGRAM:**

import java.util.Scanner;

class CountingSort {

public static void countingSort(int[] arr, int n) {

int max = arr[0];

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

if (arr[i] > max) {

max = arr[i];

}

}

int[] count = new int[max + 1];

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

count[arr[i]]++;

}

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

count[i] += count[i - 1];

}

int[] output = new int[n];

for (int i = n - 1; i >= 0; i--) {

output[count[arr[i]] - 1] = arr[i];

count[arr[i]]--;

}

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

arr[i] = output[i];

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

int[] arr = new int[n];

System.out.println("Enter " + n + " elements:");

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

arr[i] = sc.nextInt();

}

countingSort(arr, n);

System.out.println("Sorted Array:");

for (int num : arr) {

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

}

System.out.println();

}

}

**6.b**

**PROGRAM:**

import java.util.Scanner;

class RadixSort {

public static void countingSortByDigit(int[] arr, int n, int exp) {

int[] output = new int[n];

int[] count = new int[10];

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

int digit = (arr[i] / exp) % 10;

count[digit]++;

}

for (int i = 1; i < 10; i++) {

count[i] += count[i - 1];

}

for (int i = n - 1; i >= 0; i--) {

int digit = (arr[i] / exp) % 10;

output[count[digit] - 1] = arr[i];

count[digit]--;

}

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

arr[i] = output[i];

}

}

public static void radixSort(int[] arr, int n) {

int max = arr[0];

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

if (arr[i] > max) {

max = arr[i];

}

}

for (int exp = 1; max / exp > 0; exp \*= 10) {

countingSortByDigit(arr, n, exp);

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

int[] arr = new int[n];

System.out.println("Enter " + n + " elements:");

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

arr[i] = sc.nextInt();

}

radixSort(arr, n);

System.out.println("Sorted Array:");

for (int num : arr) {

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

}

System.out.println();

}

}

**6.c**

**PROGRAM:**

import java.util.Scanner;

class HeapSort {

public static void heapify(int[] arr, int n, int i) {

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < n && arr[left] > arr[largest]) {

largest = left;

}

if (right < n && arr[right] > arr[largest]) {

largest = right;

}

if (largest != i) {

int temp = arr[i];

arr[i] = arr[largest];

arr[largest] = temp;

heapify(arr, n, largest);

}

}

public static void heapSort(int[] arr, int n) {

for (int i = n / 2 - 1; i >= 0; i--) {

heapify(arr, n, i);

}

for (int i = n - 1; i > 0; i--) {

int temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

heapify(arr, i, 0);

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

int[] arr = new int[n];

System.out.println("Enter " + n + " elements:");

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

arr[i] = sc.nextInt();

}

heapSort(arr, n);

System.out.println("Sorted Array:");

for (int num : arr) {

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

}

System.out.println();

}

}

**Task 7: Introduction, tree traversal (in-order, pre-order, post-order), Binary search tree**

**7.a:**

**PROGRAM:**

import java.util.\*;

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int val) { this.val = val; }

}

public class LongestUnivaluePath {

int maxLength = 0;

public int longestUnivaluePath(TreeNode root) {

helper(root);

return maxLength;

}

private int helper(TreeNode node) {

if (node == null) return 0;

int left = helper(node.left);

int right = helper(node.right);

int leftPath = 0, rightPath = 0;

if (node.left != null && node.left.val == node.val) {

leftPath = left + 1;

}

if (node.right != null && node.right.val == node.val) {

rightPath = right + 1;

}

maxLength = Math.max(maxLength, leftPath + rightPath);

return Math.max(leftPath, rightPath);

}

public static TreeNode buildTree(Scanner sc) {

System.out.println("Enter node values in level order (use 'null' for no node):");

String val = sc.next();

if (val.equals("null")) return null;

TreeNode root = new TreeNode(Integer.parseInt(val));

Queue<TreeNode> queue = new LinkedList<>();

queue.add(root);

while (!queue.isEmpty()) {

TreeNode current = queue.poll();

if (!sc.hasNext()) break;

String leftVal = sc.next();

if (!leftVal.equals("null")) {

current.left = new TreeNode(Integer.parseInt(leftVal));

queue.add(current.left);

}

if (!sc.hasNext()) break;

String rightVal = sc.next();

if (!rightVal.equals("null")) {

current.right = new TreeNode(Integer.parseInt(rightVal));

queue.add(current.right);

}

}

return root;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

TreeNode root = buildTree(sc);

LongestUnivaluePath solution = new LongestUnivaluePath();

int result = solution.longestUnivaluePath(root);

System.out.println("Longest Univalue Path Length = " + result);

sc.close();

}

}

**7.b:**

**PROGRAM:**

import java.util.\*;

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int val) { this.val = val; }

}

public class PathSumCounter {

private int count = 0;

private int targetSum;

public int pathSum(TreeNode root, int targetSum) {

this.targetSum = targetSum;

Map<Long, Integer> prefixSumMap = new HashMap<>();

prefixSumMap.put(0L, 1);

dfs(root, 0, prefixSumMap);

return count;

}

private void dfs(TreeNode node, long currentSum, Map<Long, Integer> prefixSumMap) {

if (node == null) return;

currentSum += node.val;

count += prefixSumMap.getOrDefault(currentSum - targetSum, 0);

prefixSumMap.put(currentSum, prefixSumMap.getOrDefault(currentSum, 0) + 1);

dfs(node.left, currentSum, prefixSumMap);

dfs(node.right, currentSum, prefixSumMap);

prefixSumMap.put(currentSum, prefixSumMap.get(currentSum) - 1);

}

public static TreeNode buildTree(Scanner sc) {

System.out.println("Enter node values in level order (use 'null' for empty nodes):");

String line = sc.nextLine();

String[] vals = line.split("\\s+");

if (vals.length == 0 || vals[0].equals("null")) return null;

TreeNode root = new TreeNode(Integer.parseInt(vals[0]));

Queue<TreeNode> queue = new LinkedList<>();

queue.add(root);

int index = 1;

while (!queue.isEmpty() && index < vals.length) {

TreeNode current = queue.poll();

if (index < vals.length && !vals[index].equals("null")) {

current.left = new TreeNode(Integer.parseInt(vals[index]));

queue.add(current.left);

}

index++;

if (index < vals.length && !vals[index].equals("null")) {

current.right = new TreeNode(Integer.parseInt(vals[index]));

queue.add(current.right);

}

index++;

}

return root;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

TreeNode root = buildTree(sc);

System.out.print("Enter target sum: ");

int targetSum = sc.nextInt();

PathSumCounter solution = new PathSumCounter();

int result = solution.pathSum(root, targetSum);

System.out.println("Number of paths with sum " + targetSum + " = " + result);

sc.close();

}

}

**7.c:**

**PROGRAM:**

import java.util.\*;

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int val) { this.val = val; }

}

public class LevelOrderTraversal {

public static TreeNode buildTree(Scanner sc) {

System.out.println("Enter node values in level order (use 'null' for no node):");

String line = sc.nextLine();

String[] vals = line.split("\\s+");

if (vals.length == 0 || vals[0].equals("null")) return null;

TreeNode root = new TreeNode(Integer.parseInt(vals[0]));

Queue<TreeNode> queue = new LinkedList<>();

queue.add(root);

int index = 1;

while (!queue.isEmpty() && index < vals.length) {

TreeNode current = queue.poll();

if (index < vals.length && !vals[index].equals("null")) {

current.left = new TreeNode(Integer.parseInt(vals[index]));

queue.add(current.left);

}

index++;

if (index < vals.length && !vals[index].equals("null")) {

current.right = new TreeNode(Integer.parseInt(vals[index]));

queue.add(current.right);

}

index++;

}

return root;

}

public static List<List<Integer>> levelOrder(TreeNode root) {

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

if (root == null) return result;

Queue<TreeNode> queue = new LinkedList<>();

queue.add(root);

while (!queue.isEmpty()) {

int levelSize = queue.size();

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

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

TreeNode current = queue.poll();

currentLevel.add(current.val);

if (current.left != null) queue.add(current.left);

if (current.right != null) queue.add(current.right);

}

result.add(currentLevel);

}

return result;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

TreeNode root = buildTree(sc);

List<List<Integer>> traversal = levelOrder(root);

System.out.println("Level Order Traversal: " + traversal);

sc.close();

}

}

**Task 8: Graph - Introduction Adjacency Matrix and List, Depth First search, Breadth First Search, Manacher’s algorithm.**

**8.a:**

**PROGRAM:**

import java.util.\*;

public class CheapestFlight {

static class Flight {

int dest, cost;

Flight(int d, int c) { dest = d; cost = c; }

}

public static int findCheapestPrice(int n, int[][] flights, int src, int dst, int K) {

List<List<Flight>> graph = new ArrayList<>();

for (int i = 0; i < n; i++) graph.add(new ArrayList<>());

for (int[] f : flights)

graph.get(f[0]).add(new Flight(f[1], f[2]));

PriorityQueue<int[]> pq = new PriorityQueue<>(Comparator.comparingInt(a -> a[0]));

pq.offer(new int[]{0, src, 0});

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

while (!pq.isEmpty()) {

int[] cur = pq.poll();

int cost = cur[0], city = cur[1], stops = cur[2];

if (city == dst) return cost;

if (visited.containsKey(city) && visited.get(city) <= stops) continue;

visited.put(city, stops);

if (stops <= K) {

for (Flight f : graph.get(city)) {

pq.offer(new int[]{cost + f.cost, f.dest, stops + 1});

}

}

}

return -1;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt(); // number of cities

int m = sc.nextInt(); // number of flights

int[][] flights = new int[m][3];

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

flights[i][0] = sc.nextInt(); // from city

flights[i][1] = sc.nextInt(); // to city

flights[i][2] = sc.nextInt(); // cost

}

int src = sc.nextInt();

int dst = sc.nextInt();

int K = sc.nextInt();

System.out.println(findCheapestPrice(n, flights, src, dst, K));

sc.close();

}

}

**8.b:**

**PROGRAM:**

import java.util.\*;

public class MinCostConnectGroups {

public static int connectTwoGroups(List<List<Integer>> cost) {

int n = cost.size(), m = cost.get(0).size();

int[] minCost2 = new int[m];

Arrays.fill(minCost2, Integer.MAX\_VALUE);

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

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

minCost2[j] = Math.min(minCost2[j], cost.get(i).get(j));

}

}

int maxMask = 1 << m;

int[] dp = new int[maxMask];

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

dp[0] = 0;

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

int[] newDp = new int[maxMask];

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

for (int mask = 0; mask < maxMask; mask++) {

if (dp[mask] == Integer.MAX\_VALUE) continue;

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

int newMask = mask | (1 << j);

int costToAdd = cost.get(i).get(j);

if (newDp[newMask] > dp[mask] + costToAdd)

newDp[newMask] = dp[mask] + costToAdd;

}

}

dp = newDp;

}

int ans = Integer.MAX\_VALUE;

for (int mask = 0; mask < maxMask; mask++) {

int sum = dp[mask];

if (sum == Integer.MAX\_VALUE) continue;

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

if ((mask & (1 << j)) == 0) {

sum += minCost2[j];

}

}

ans = Math.min(ans, sum);

}

return ans;

}

public static void main(String[] args) {

List<List<Integer>> cost = Arrays.asList(

Arrays.asList(15, 96),

Arrays.asList(36, 2)

);

int result = connectTwoGroups(cost);

System.out.println("Minimum cost = " + result);

}

}

**8.c:**

**PROGRAM:**

import java.util.\*;

public class DecodeString {

public static String decodeString(String s) {

Stack<Integer> countStack = new Stack<>();

Stack<StringBuilder> stringStack = new Stack<>();

StringBuilder currentString = new StringBuilder();

int currentNumber = 0;

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

if (Character.isDigit(c)) {

currentNumber = currentNumber \* 10 + (c - '0');

} else if (c == '[') {

countStack.push(currentNumber);

stringStack.push(currentString);

currentNumber = 0;

currentString = new StringBuilder();

} else if (c == ']') {

int repeat = countStack.pop();

StringBuilder temp = currentString;

currentString = stringStack.pop();

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

currentString.append(temp);

}

} else {

currentString.append(c);

}

}

return currentString.toString();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

String encoded = sc.nextLine();

String decoded = decodeString(encoded);

System.out.println("Decoded Output: " + decoded);

sc.close();

}

}

**Task 9: Backtracking-Introduction, Backtrack vs Divide and Conquer**

**9.a:**

**PROGRAM:**

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class LetterCombinationsPhoneNumber {

private static final String[] KEYPAD = {

"",

"",

"abc",

"def",

"ghi",

"jkl",

"mno",

"pqrs",

"tuv",

"wxyz"

};

public static List<String> letterCombinations(String digits) {

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

if (digits == null || digits.length() == 0) {

return result;

}

backtrack(digits, 0, new StringBuilder(), result);

return result;

}

private static void backtrack(String digits, int index, StringBuilder current, List<String> result) {

if (index == digits.length()) {

result.add(current.toString());

return;

}

String letters = KEYPAD[digits.charAt(index) - '0'];

for (char letter : letters.toCharArray()) {

current.append(letter);

backtrack(digits, index + 1, current, result);

current.deleteCharAt(current.length() - 1);

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter digits (2-9): ");

String input = scanner.nextLine().trim();

if (!input.matches("[2-9]+")) {

System.out.println("Invalid input! Please enter digits between 2 and 9 only.");

scanner.close();

return;

}

List<String> output = letterCombinations(input);

System.out.println("Output: " + output);

scanner.close();

}

}

**9.b:**

**PROGRAM:**

import java.util.Scanner;

public class RegexMatcher {

public static boolean isMatch(String s, String p) {

if (p.isEmpty()) {

return s.isEmpty();

}

boolean firstMatch = (!s.isEmpty() &&

(s.charAt(0) == p.charAt(0) || p.charAt(0) == '.'));

if (p.length() >= 2 && p.charAt(1) == '\*') {

return (isMatch(s, p.substring(2)) ||

(firstMatch && isMatch(s.substring(1), p)));

} else {

return firstMatch && isMatch(s.substring(1), p.substring(1));

}

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter input string (s): ");

String s = sc.nextLine();

System.out.print("Enter pattern (p): ");

String p = sc.nextLine();

boolean matched = isMatch(s, p);

System.out.println("Does the string match the pattern? " + matched);

sc.close();

}

}

**9.c:**

**PROGRAM:**

import java.util.\*;

public class SequentialDigits {

public static List<Integer> sequentialDigits(int low, int high) {

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

int lowLen = String.valueOf(low).length();

int highLen = String.valueOf(high).length();

for (int len = lowLen; len <= highLen; len++) {

for (int start = 1; start <= 9; start++) {

backtrack(low, high, len, start, 0, result);

}

}

Collections.sort(result);

return result;

}

private static void backtrack(int low, int high, int len, int currentDigit, int currentNum, List<Integer> result) {

if (len == 0) {

if (currentNum >= low && currentNum <= high) {

result.add(currentNum);

}

return;

}

if (currentDigit > 9) return;

currentNum = currentNum \* 10 + currentDigit;

backtrack(low, high, len - 1, currentDigit + 1, currentNum, result);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter low: ");

int low = sc.nextInt();

System.out.print("Enter high: ");

int high = sc.nextInt();

List<Integer> res = sequentialDigits(low, high);

System.out.println("Sequential digits in range: " + res);

sc.close();

}

}