**DAY-3 PROGRAMS**

**21.** **#include <stdio.h>**

**// Function to find the maximum of two integers**

**int max(int a, int b) {**

**return (a > b) ? a : b;**

**}**

**// Function to solve the Knapsack problem using dynamic programming**

**int knapSack(int W, int wt[], int val[], int n) {**

**int i, w;**

**int K[n + 1][W + 1];**

**// Build table K[][] in a bottom-up manner**

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

**for (w = 0; w <= W; w++) {**

**if (i == 0 || w == 0)**

**K[i][w] = 0;**

**else if (wt[i - 1] <= w)**

**K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w]);**

**else**

**K[i][w] = K[i - 1][w];**

**}**

**}**

**// Return the maximum value that can be put in a knapsack of capacity W**

**return K[n][W];**

**}**

**int main() {**

**int val[] = {60, 100, 120};**

**int wt[] = {10, 20, 30};**

**int W = 50;**

**int n = sizeof(val) / sizeof(val[0]);**

**printf("Maximum value in Knapsack = %d\n", knapSack(W, wt, val, n));**

**return 0;**

**}**

**22.** **#include <stdio.h>**

**#include <limits.h>**

**// Function to calculate the optimal cost**

**void optimalBST(int keys[], int freq[], int n) {**

**int cost[n][n]; // cost[i][j] will store the cost of the optimal BST that can be built from keys[i] to keys[j]**

**// Initialize cost when there is only one key in the range**

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

**cost[i][i] = freq[i];**

**}**

**// Compute the optimal cost for larger subproblems**

**for (int length = 2; length <= n; length++) { // length is the number of keys in the subproblem**

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

**int j = i + length - 1;**

**cost[i][j] = INT\_MAX;**

**// Calculate the sum of frequencies from i to j**

**int freqSum = 0;**

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

**freqSum += freq[k];**

**}**

**// Try making each key in the subproblem root and calculate the cost**

**for (int r = i; r <= j; r++) {**

**int c = ((r > i) ? cost[i][r - 1] : 0) + // cost of left subtree**

**((r < j) ? cost[r + 1][j] : 0) + // cost of right subtree**

**freqSum; // cost of the root (sum of frequencies in the range)**

**if (c < cost[i][j]) {**

**cost[i][j] = c;**

**}**

**}**

**}**

**}**

**printf("The cost of the optimal binary search tree is %d\n", cost[0][n - 1]);**

**}**

**int main() {**

**int keys[] = {10, 12, 20};**

**int freq[] = {34, 8, 50};**

**int n = sizeof(keys) / sizeof(keys[0]);**

**optimalBST(keys, freq, n);**

**return 0;**

**}**

**23.** **#include <stdio.h>**

**// Function to compute binomial coefficient C(n, k)**

**int binomialCoefficient(int n, int k) {**

**int C[n + 1][k + 1];**

**int i, j;**

**// Initialize all elements of the array to 0**

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

**for (j = 0; j <= k; j++) {**

**C[i][j] = 0;**

**}**

**}**

**// Compute the binomial coefficients**

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

**for (j = 0; j <= (i < k ? i : k); j++) {**

**// Base cases: C(i, 0) = 1 and C(i, i) = 1**

**if (j == 0 || j == i) {**

**C[i][j] = 1;**

**} else {**

**C[i][j] = C[i - 1][j - 1] + C[i - 1][j];**

**}**

**}**

**}**

**return C[n][k];**

**}**

**int main() {**

**int n, k;**

**printf("Enter n and k: ");**

**scanf("%d %d", &n, &k);**

**printf("C(%d, %d) = %d\n", n, k, binomialCoefficient(n, k));**

**return 0;**

**}**

**24.** #include <stdio.h>

// Function to reverse the digits of a number

int reverseNumber(int num, int rev) {

if (num == 0) {

return rev;

}

rev = (rev \* 10) + (num % 10);

return reverseNumber(num / 10, rev);

}

int main() {

int num, reversedNum;

printf("Enter a number: ");

scanf("%d", &num);

reversedNum = reverseNumber(num, 0);

printf("Reversed Number: %d\n", reversedNum);

return 0;

}

**25#include <stdio.h>**

**// Function to check if a number is a perfect number**

**int isPerfectNumber(int num) {**

**int sum = 0;**

**// Find all divisors and add them**

**for (int i = 1; i <= num / 2; i++) {**

**if (num % i == 0) {**

**sum += i;**

**}**

**}**

**// Check if the sum of divisors is equal to the number**

**return (sum == num);**

**}**

**int main() {**

**int num;**

**printf("Enter a number: ");**

**scanf("%d", &num);**

**if (isPerfectNumber(num)) {**

**printf("%d is a perfect number.\n", num);**

**} else {**

**printf("%d is not a perfect number.\n", num);**

**}**

**return 0;**

**}**

**26.** **#include <stdio.h>**

**#include <limits.h>**

**#define MAX 20**

**#define INF INT\_MAX**

**int n, dist[MAX][MAX], dp[MAX][1 << MAX];**

**int tsp(int mask, int pos) {**

**if (mask == (1 << n) - 1) return dist[pos][0]; // Return to start**

**if (dp[pos][mask] != -1) return dp[pos][mask];**

**int answer = INF;**

**for (int city = 0; city < n; city++) {**

**if (!(mask & (1 << city))) { // If city is not visited**

**int new\_answer = dist[pos][city] + tsp(mask | (1 << city), city);**

**if (new\_answer < answer) answer = new\_answer;**

**}**

**}**

**return dp[pos][mask] = answer;**

**}**

**int main() {**

**printf("Enter the number of cities: ");**

**scanf("%d", &n);**

**printf("Enter the distance matrix:\n");**

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

**for (int j = 0; j < n; j++)**

**scanf("%d", &dist[i][j]);**

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

**for (int j = 0; j < (1 << n); j++)**

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

**printf("The minimum cost to visit all cities is: %d\n", tsp(1, 0));**

**return 0;**

**}**

**27.** **#include <stdio.h>**

**void printNumbers(int n) {**

**if (n == 0)**

**return;**

**printNumbers(n - 1); // Recursive call to print the previous line**

**// Print numbers from 1 to n**

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

**printf("%d ", i);**

**}**

**printf("\n"); // Move to the next line after printing numbers**

**}**

**void printPattern(int rows) {**

**printNumbers(rows);**

**}**

**int main() {**

**int rows = 4;**

**printPattern(rows);**

**return 0;**

**}**

**28.** **#include <stdio.h>**

**void printNumbers(int n) {**

**if (n == 0)**

**return;**

**printNumbers(n - 1); // Recursive call to print the previous line**

**// Print numbers from 1 to n**

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

**printf("%d ", i);**

**}**

**printf("\n"); // Move to the next line after printing numbers**

**}**

**void printPattern() {**

**int rows = 4; // Set number of rows to 4**

**printNumbers(rows);**

**}**

**int main() {**

**printPattern(); // Call the function to print the pattern**

**return 0;**

**}**

**29.** **#include <stdio.h>**

**void printRow(int start, int count) {**

**if (count == 0)**

**return;**

**printf("%d ", start);**

**printRow(start + 1, count - 1);**

**}**

**void printFloydsTriangle(int rows, int currentRow, int \*num) {**

**if (currentRow > rows)**

**return;**

**printRow(\*num, currentRow);**

**\*num += currentRow; // Update the starting number for the next row**

**printf("\n");**

**printFloydsTriangle(rows, currentRow + 1, num);**

**}**

**int main() {**

**int rows = 4;**

**int num = 1; // Starting number for Floyd's Triangle**

**printFloydsTriangle(rows, 1, &num);**

**return 0;**

**}**

**30.** **#include <stdio.h>**

**// Function to calculate the factorial of a number**

**int factorial(int n) {**

**if (n == 0 || n == 1)**

**return 1;**

**return n \* factorial(n - 1);**

**}**

**// Function to calculate binomial coefficient**

**int binomialCoefficient(int n, int k) {**

**return factorial(n) / (factorial(k) \* factorial(n - k));**

**}**

**// Function to print a single row of Pascal's Triangle**

**void printRow(int row) {**

**for (int i = 0; i <= row; i++) {**

**printf("%d ", binomialCoefficient(row, i));**

**}**

**printf("\n");**

**}**

**// Recursive function to print Pascal's Triangle**

**void printPascalsTriangle(int rows) {**

**if (rows < 0)**

**return;**

**printPascalsTriangle(rows - 1); // Recursively print previous rows**

**printRow(rows); // Print the current row**

**}**

**int main() {**

**int rows = 4; // Number of rows to print**

**printPascalsTriangle(rows - 1);**

**return 0;**

**}**