DAY-3 PROGRAMS:

1.Write a program to find reverse of a given number.

Program:

#include <stdio.h>

int main() {

int number, reversedNumber = 0;

printf("Enter a number: ");

scanf("%d", &number);

int originalNumber = number;

while (number != 0) {

int digit = number % 10;

reversedNumber = reversedNumber \* 10 + digit;

number /= 10;

}

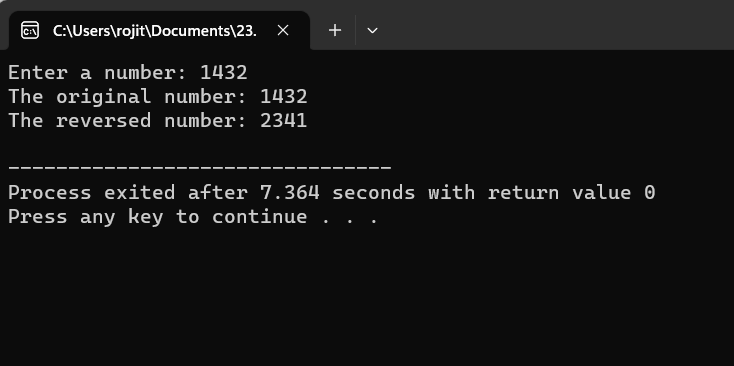
printf("The original number: %d\n", originalNumber);

printf("The reversed number: %d\n", reversedNumber);

return 0;

}

Output:



2.Write a program to find the Perfect number:

Program:

#include <stdio.h>

int isPerfectNumber(int num)

{

int sum = 0;

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

{

if (num % i == 0) {

sum += i;

}

}

return sum == num;

}

int main()

{

int number;

printf("Enter a number: ");

scanf("%d", &number);

if (isPerfectNumber(number))

{

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

}

Else

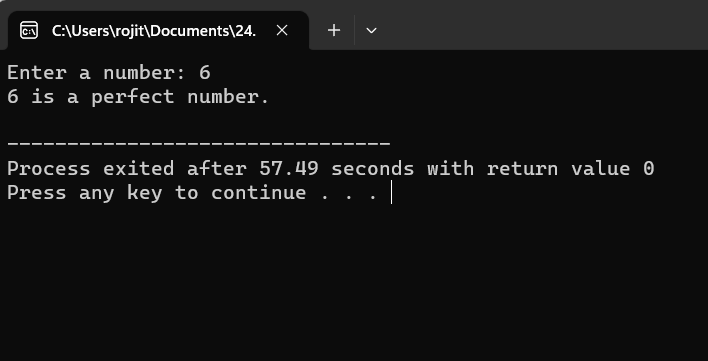
{

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

}

return 0;

}

Output:

3.Write a program to perform Travelling Salesperson Problem using Dynamic programming:

Program:

#include <stdio.h>

#include <limits.h>

#define MAX\_CITIES 10

int min(int a, int b) {

return (a < b) ? a : b;

}

int tsp(int n, int graph[MAX\_CITIES][MAX\_CITIES], int mask, int pos, int dp[MAX\_CITIES][1 << MAX\_CITIES]) {

if (mask == (1 << n) - 1) {

return graph[pos][0];

}

if (dp[pos][mask] != -1) {

return dp[pos][mask];

}

int ans = INT\_MAX;

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

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

int newAns = graph[pos][i] + tsp(n, graph, mask | (1 << i), i, dp);

ans = min(ans, newAns);

}

}

return dp[pos][mask] = ans;

}

int main() {

int n, i, j;

printf("Enter the number of cities (maximum %d): ", MAX\_CITIES);

scanf("%d", &n);

int graph[MAX\_CITIES][MAX\_CITIES];

int dp[MAX\_CITIES][1 << MAX\_CITIES];

printf("Enter the adjacency matrix for the graph:\n");

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

for (j = 0; j < n; j++) {

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

}

}

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

for (j = 0; j < (1 << n); j++) {

dp[i][j] = -1;

}

}

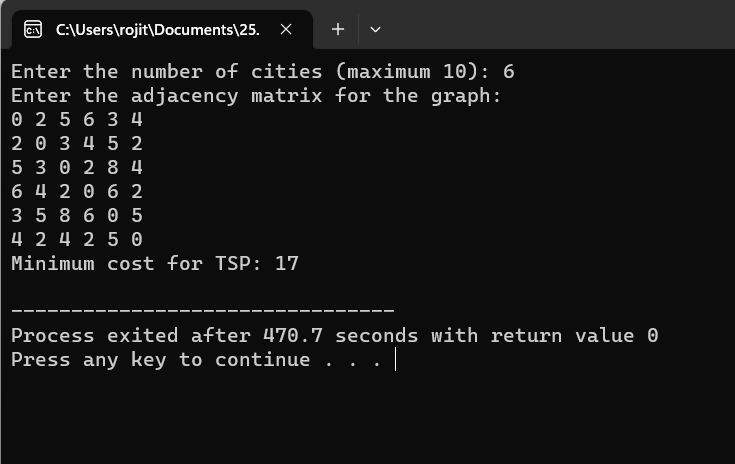
int minCost = tsp(n, graph, 1, 0, dp);

printf("Minimum cost for TSP: %d\n", minCost);

return 0;

}

Output:



4.Write a program for the given pattern:

Program:

#include <stdio.h>

int main() {

int n;

printf("Enter the value of n: ");

scanf("%d", &n);

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

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

printf("%d ", j);

}

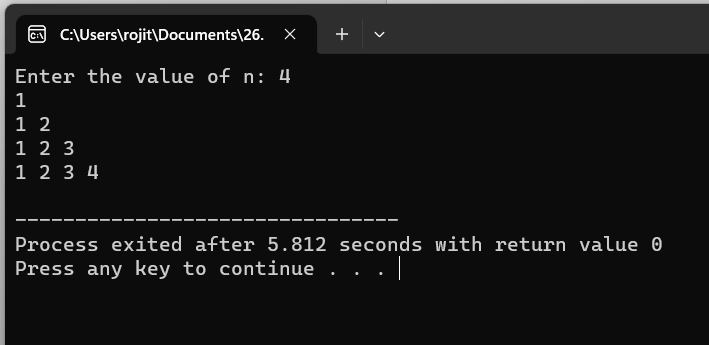
printf("\n");

}

return 0;

}

Output:



5. Write a program to perform Floyd’s algorithm:

Program:

#include <stdio.h>

#define INF 9999

#define MAX\_NODES 10

void floyd(int graph[MAX\_NODES][MAX\_NODES], int n) {

int distance[MAX\_NODES][MAX\_NODES];

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

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

distance[i][j] = graph[i][j];

}

}

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

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

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

if (distance[i][k] + distance[k][j] < distance[i][j]) {

distance[i][j] = distance[i][k] + distance[k][j];

}

}

}

}

printf("Shortest paths between all pairs of nodes:\n");

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

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

if (distance[i][j] == INF) {

printf("INF\t");

} else {

printf("%d\t", distance[i][j]);

}

}

printf("\n");

}

}

int main() {

int n;

printf("Enter the number of nodes in the graph: ");

scanf("%d", &n);

int graph[MAX\_NODES][MAX\_NODES];

printf("Enter the adjacency matrix (use INF for no direct edge):\n");

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

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

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

if (graph[i][j] == -1) {

graph[i][j] = INF;

}

}

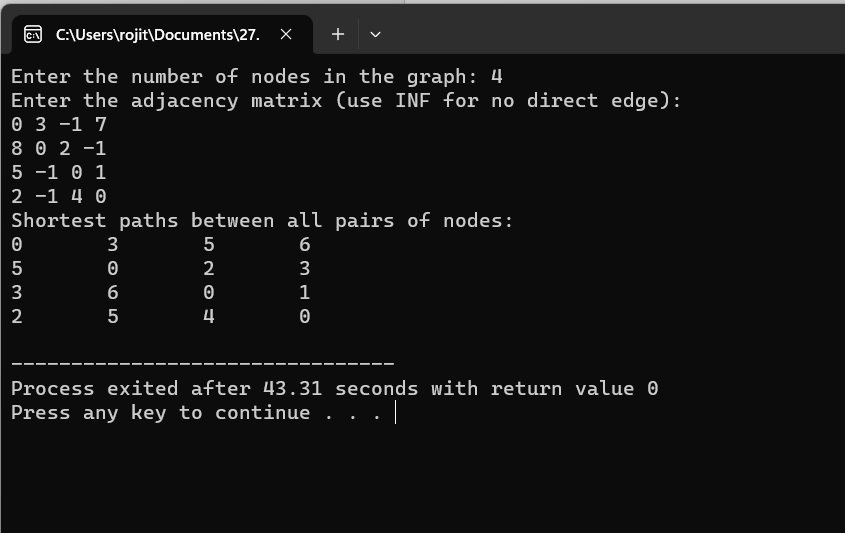
}

floyd(graph, n);

return 0;

}

Output:



6.Write a program for pascal triangle:

Program:

#include <stdio.h>

int factorial(int n) {

if (n <= 1)

return 1;

else

return n \* factorial(n - 1);

}

int main() {

int rows;

printf("Enter the number of rows for Pascal's Triangle: ");

scanf("%d", &rows);

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

for (int space = 0; space < rows - i; space++) {

printf(" ");

}

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

int coefficient = factorial(i) / (factorial(j) \* factorial(i - j));

printf("%d ", coefficient);

}

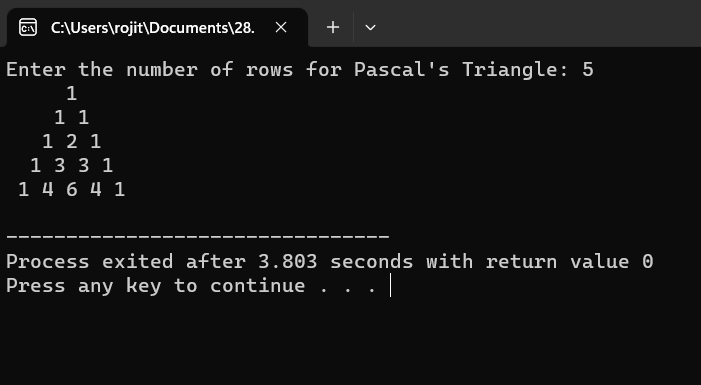
printf("\n");

}

return 0;

}

Output:



7.Write a program to find the optimal cost by using appropriate algorithm:

Program:

#include <stdio.h>

#include <stdlib.h>

#define MAX\_NODES 20

struct Edge {

int source, destination, weight;

};

void swap(struct Edge \*a, struct Edge \*b) {

struct Edge temp = \*a;

\*a = \*b;

\*b = temp;

}

int compareEdges(const void \*a, const void \*b) {

return ((struct Edge \*)a)->weight - ((struct Edge \*)b)->weight;

}

int findParent(int parent[], int node) {

if (parent[node] == -1)

return node;

return findParent(parent, parent[node]);

}

void kruskal(int graph[MAX\_NODES][MAX\_NODES], int numNodes) {

struct Edge edges[MAX\_NODES \* MAX\_NODES];

int edgeCount = 0;

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

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

if (graph[i][j] != 0) {

edges[edgeCount].source = i;

edges[edgeCount].destination = j;

edges[edgeCount].weight = graph[i][j];

edgeCount++;

}

}

}

qsort(edges, edgeCount, sizeof(struct Edge), compareEdges);

int parent[MAX\_NODES];

for (int i = 0; i < numNodes; i++)

parent[i] = -1;

printf("Minimum Spanning Tree:\n");

int mstWeight = 0;

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

int sourceParent = findParent(parent, edges[i].source);

int destParent = findParent(parent, edges[i].destination);

if (sourceParent != destParent) {

printf("%d -- %d : %d\n", edges[i].source, edges[i].destination, edges[i].weight);

mstWeight += edges[i].weight;

parent[sourceParent] = destParent;

}

}

printf("Minimum Spanning Tree Weight: %d\n", mstWeight);

}

int main() {

int numNodes;

printf("Enter the number of nodes in the graph: ");

scanf("%d", &numNodes);

int graph[MAX\_NODES][MAX\_NODES];

printf("Enter the adjacency matrix of the graph:\n");

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

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

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

}

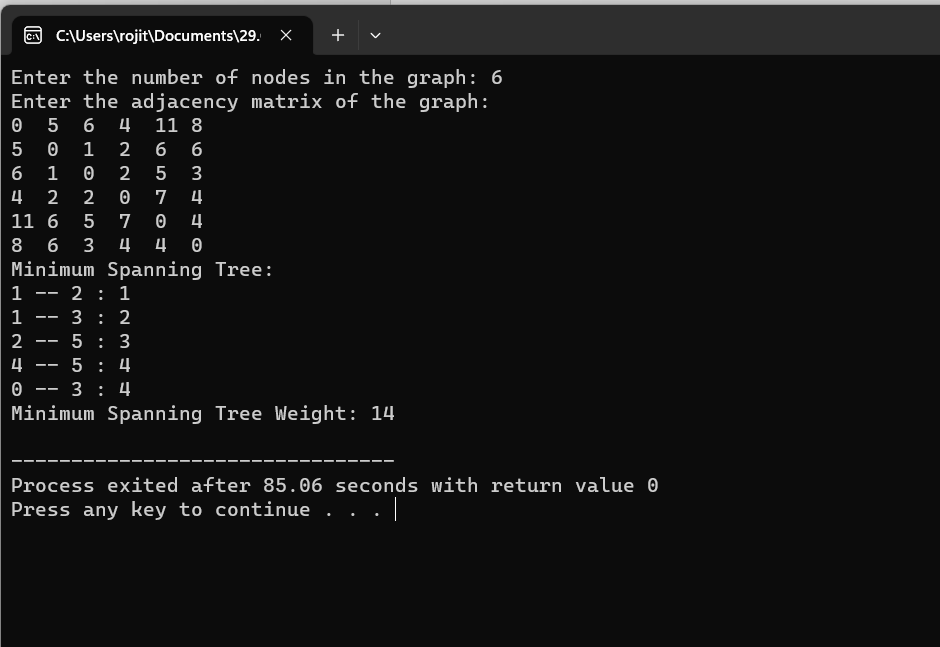
}

kruskal(graph, numNodes);

return 0;

}

Output:



8.Write a program to find the sum of digits:

Program:

#include <stdio.h>

int main() {

int num, originalNum, remainder, sum = 0;

printf("Enter an integer: ");

scanf("%d", &num);

originalNum = num;

while (num != 0) {

remainder = num % 10;

sum += remainder;

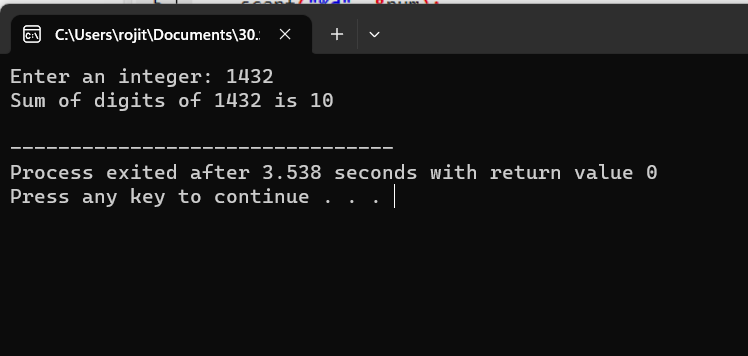
num /= 10;

}

printf("Sum of digits of %d is %d\n", originalNum, sum);

return 0;

}

Output:  


9.Write a program to print minimum and maximum value sequency for all the numbers in a list:

Program:

#include <stdio.h>

#define MAX\_ELEMENTS 100

void arrangeSequentially(int arr[], int n) {

int temp, i, j;

for (i = 0; i < n - 1; i++) {

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

if (arr[i] > arr[j]) {

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

}

int main()

{

int n, i, arr[MAX\_ELEMENTS];

printf("Enter the number of elements (maximum %d): ", MAX\_ELEMENTS);

scanf("%d", &n);

printf("Enter %d elements:\n", n);

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

{

scanf("%d", &arr[i]);

}

arrangeSequentially(arr, n);

printf("Sequentially arranged elements: ");

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

{

printf("%d ", arr[i]);

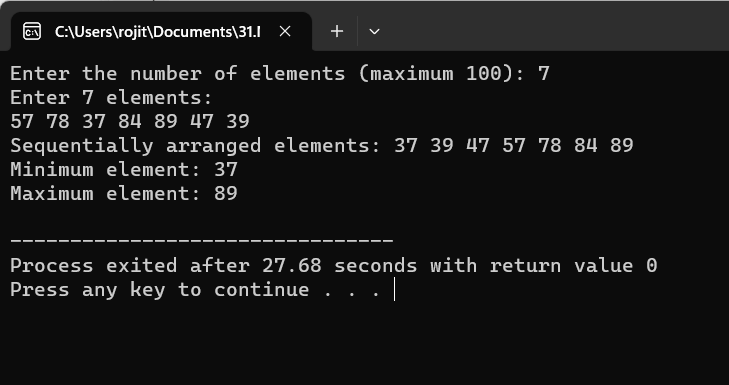
}

printf("\nMinimum element: %d\n", arr[0]);

printf("Maximum element: %d\n", arr[n - 1]);

}

Output:



10.Write a program to perform n Queens Problem using Backtracking:

Program:

#include<stdio.h>

#include<math.h>

int board[20],count;

int main()

{

int n,i,j;

void queen(int row,int n);

printf(" - N Queens Problem Using Backtracking -");

printf("\n\nEnter number of Queens:");

scanf("%d",&n);

queen(1,n);

return 0;

}

void print(int n)

{

int i,j;

printf("\n\nSolution %d:\n\n",++count);

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

printf("\t%d",i);

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

{

printf("\n\n%d",i);

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

{

if(board[i]==j)

printf("\tQ");

else

printf("\t-");

}

}

}

int place(int row,int column)

{

int i;

for(i=1;i<=row-1;++i)

{

if(board[i]==column)

return 0;

else

if(abs(board[i]-column)==abs(i-row))

return 0;

}

return 1;

}

void queen(int row,int n)

{

int column;

for(column=1;column<=n;++column)

{

if(place(row,column))

{

board[row]=column;

if(row==n)

print(n);

else

queen(row+1,n);

}

}

}

Output:

