Day-4 Programs:

1.Write a C program to insert a number in a list:

Program:

#include <stdio.h>

int main() {

int maxSize;

printf("Enter the size of the array: ");

scanf("%d", &maxSize);

int arr[maxSize];

int numElements = 0;

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

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

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

numElements++;

}

int numInsert;

printf("Enter the number of elements to insert: ");

scanf("%d", &numInsert);

printf("Enter %d numbers to insert:\n", numInsert);

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

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

numElements++;

}

printf("Array contents with inserted elements: ");

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

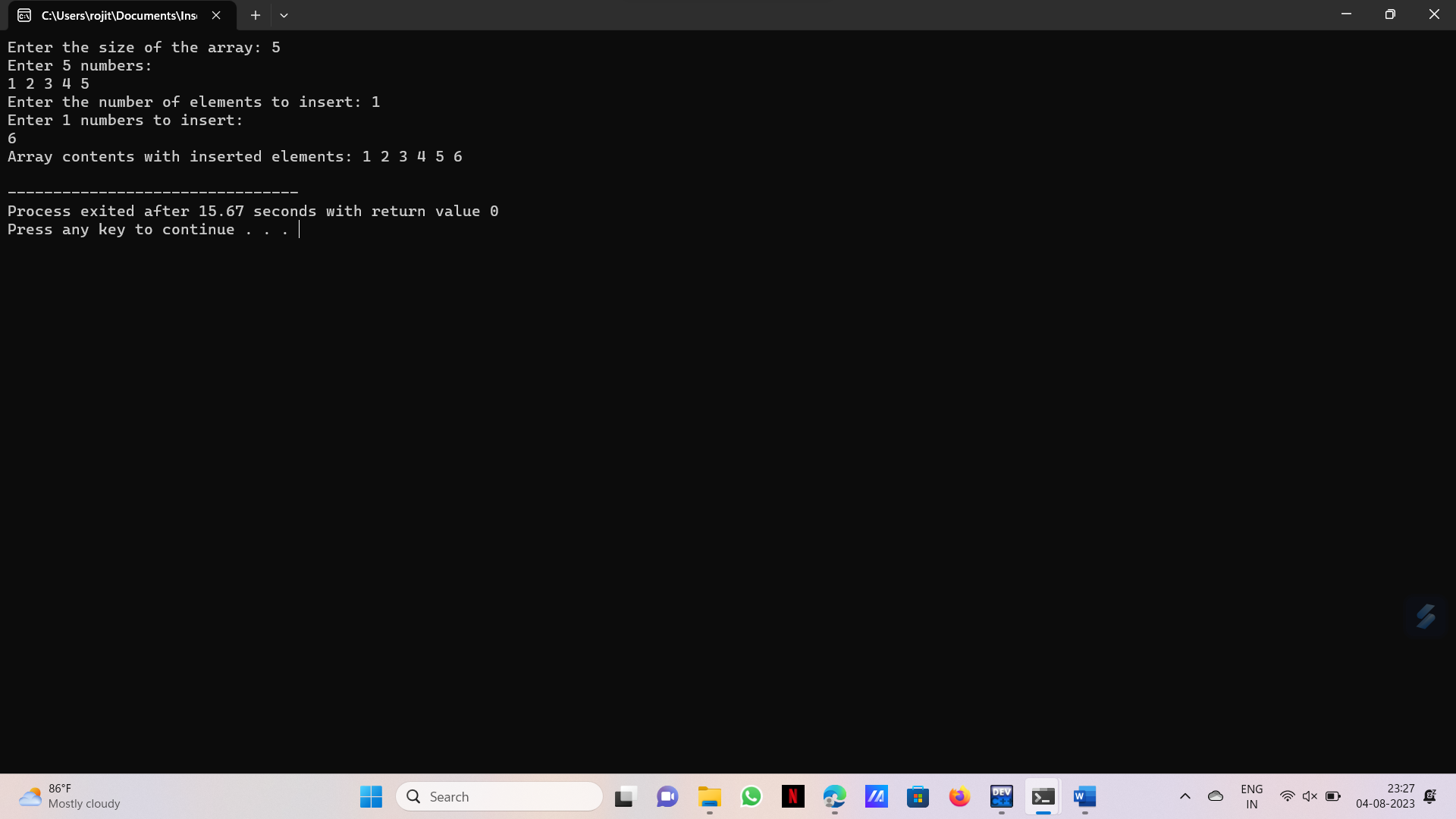
printf("%d ", arr[i]);

}

printf("\n");

}

Output:



2.Write a program to perform sum of subsets problem using backtracking:

Program:

#include <stdio.h>

#include <stdbool.h>

void generateSubsets(int arr[], int n, bool subset[], int index, int targetSum, int currentSum) {

if (index == n) {

if (currentSum == targetSum) {

printf("Subset with target sum %d: {", targetSum);

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

if (subset[i]) {

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

}

}

printf(" }\n");

}

return;

}

subset[index] = true;

generateSubsets(arr, n, subset, index + 1, targetSum, currentSum + arr[index]);

subset[index] = false;

generateSubsets(arr, n, subset, index + 1, targetSum, currentSum);

}

int main() {

int n;

printf("Enter the number of elements: ");

scanf("%d", &n);

int arr[n];

printf("Enter the elements:\n");

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

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

}

int targetSum;

printf("Enter the target sum: ");

scanf("%d", &targetSum);

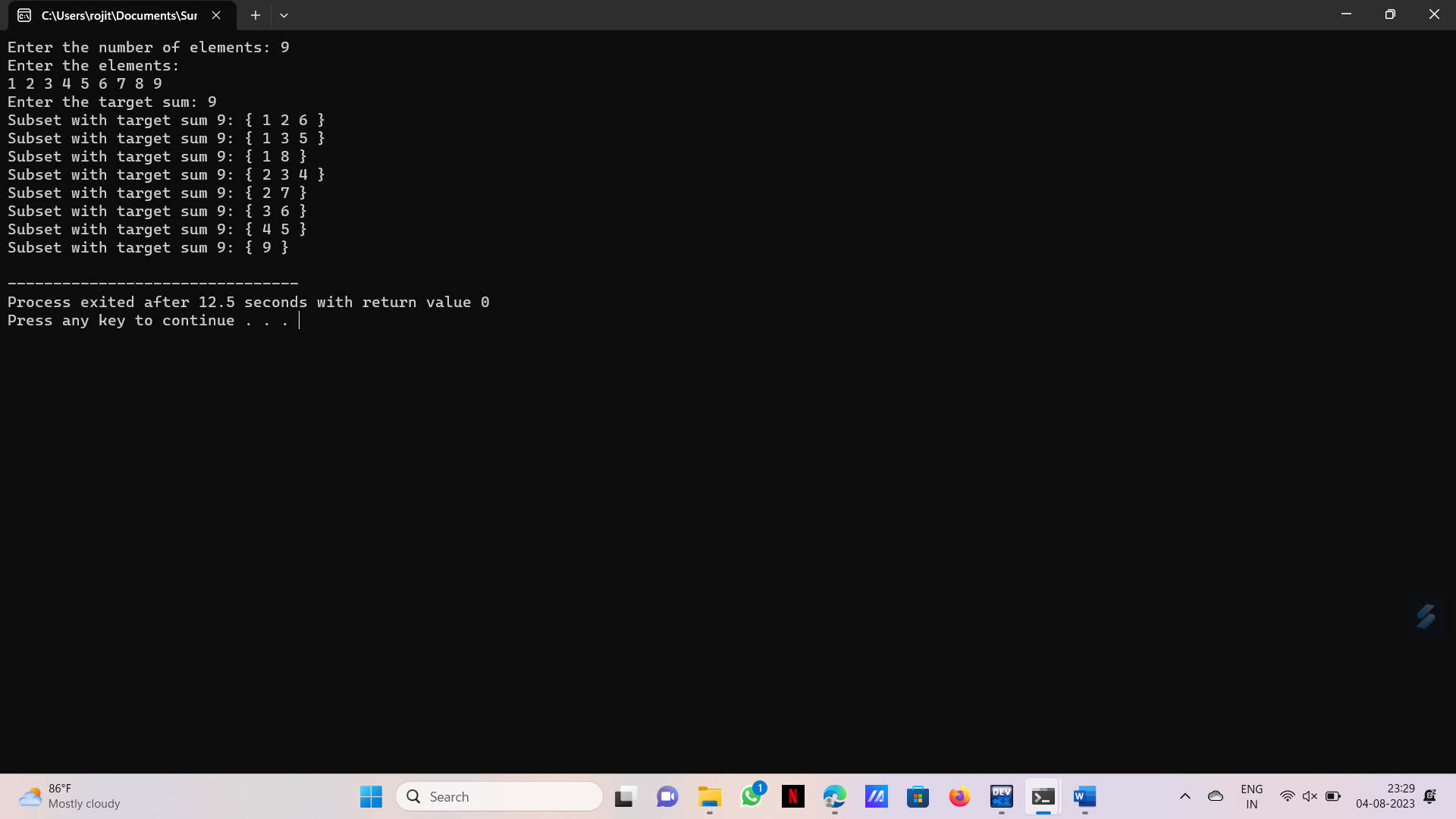
bool subset[n];

generateSubsets(arr, n, subset, 0, targetSum, 0);

return 0;

}

Output:



3.Write a Program to perform graph Colouring problem using backtracking:

Program:

#include <stdio.h>

#include <stdbool.h>

#define MAX\_VERTICES 10

#define MAX\_COLORS 10

int graph[MAX\_VERTICES][MAX\_VERTICES];

int numVertices, numColors;

int colorAssignment[MAX\_VERTICES];

char colorNames[MAX\_COLORS][20];

bool isSafe(int v, int c) {

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

if (graph[v][i] && colorAssignment[i] == c)

return false;

}

return true;

}

void printSolution() {

static int solutionCount = 0;

solutionCount++;

printf("Solution %d:\n", solutionCount);

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

printf("Vertex %d: Color %s\n", i + 1, colorNames[colorAssignment[i]]);

}

printf("\n");

}

void graphColoringUtil(int v) {

if (v == numVertices) {

printSolution();

return;

}

for (int c = 0; c < numColors; c++) {

if (isSafe(v, c)) {

colorAssignment[v] = c;

graphColoringUtil(v + 1);

colorAssignment[v] = -1; // Backtrack

}

}

}

void graphColoring() {

graphColoringUtil(0);

}

int main() {

printf("Enter the number of vertices: ");

scanf("%d", &numVertices);

printf("Enter the number of colors: ");

scanf("%d", &numColors);

printf("Enter color names:\n");

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

scanf("%s", colorNames[i]);

}

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

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

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

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

}

}

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

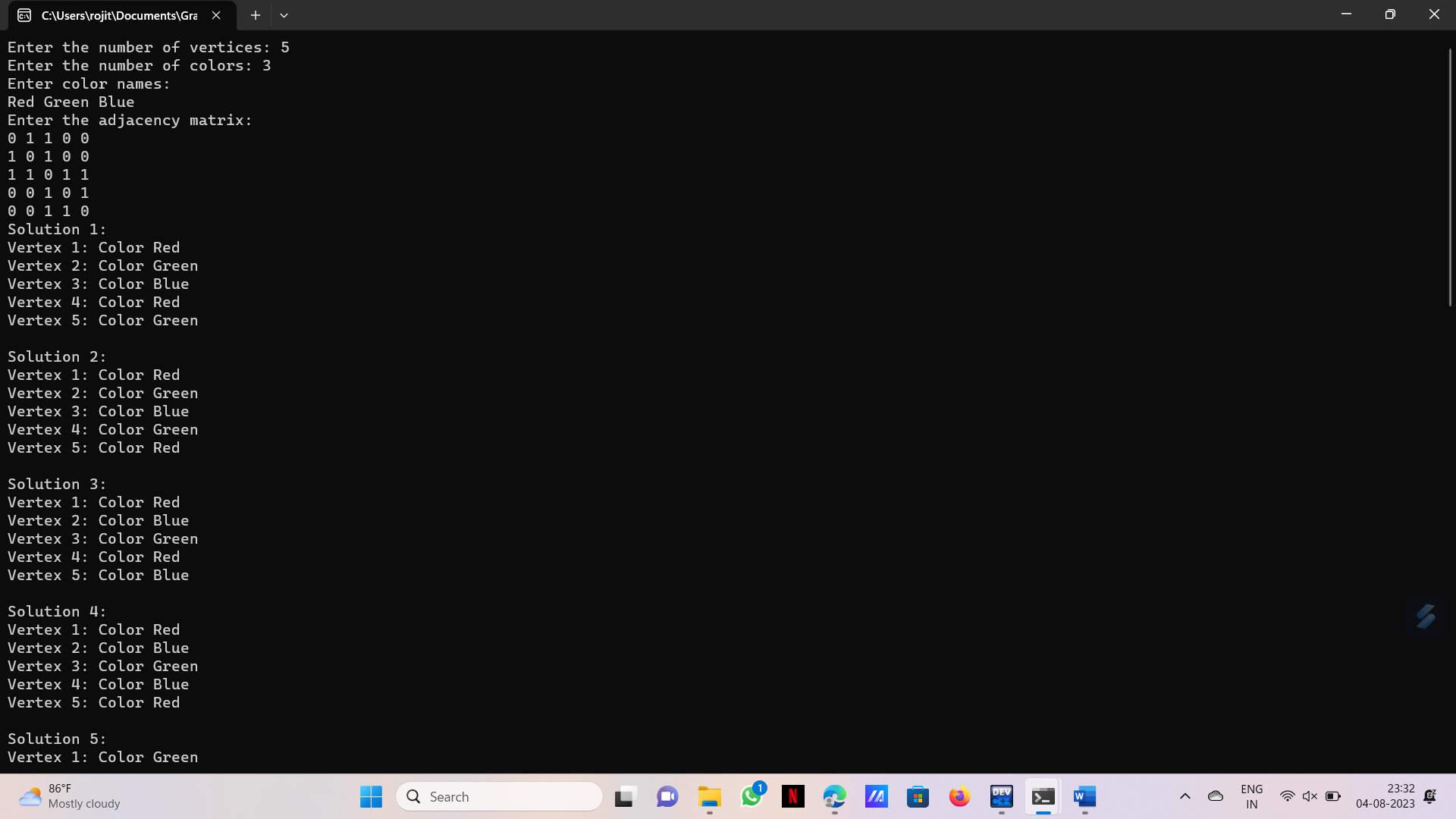
colorAssignment[i] = -1;

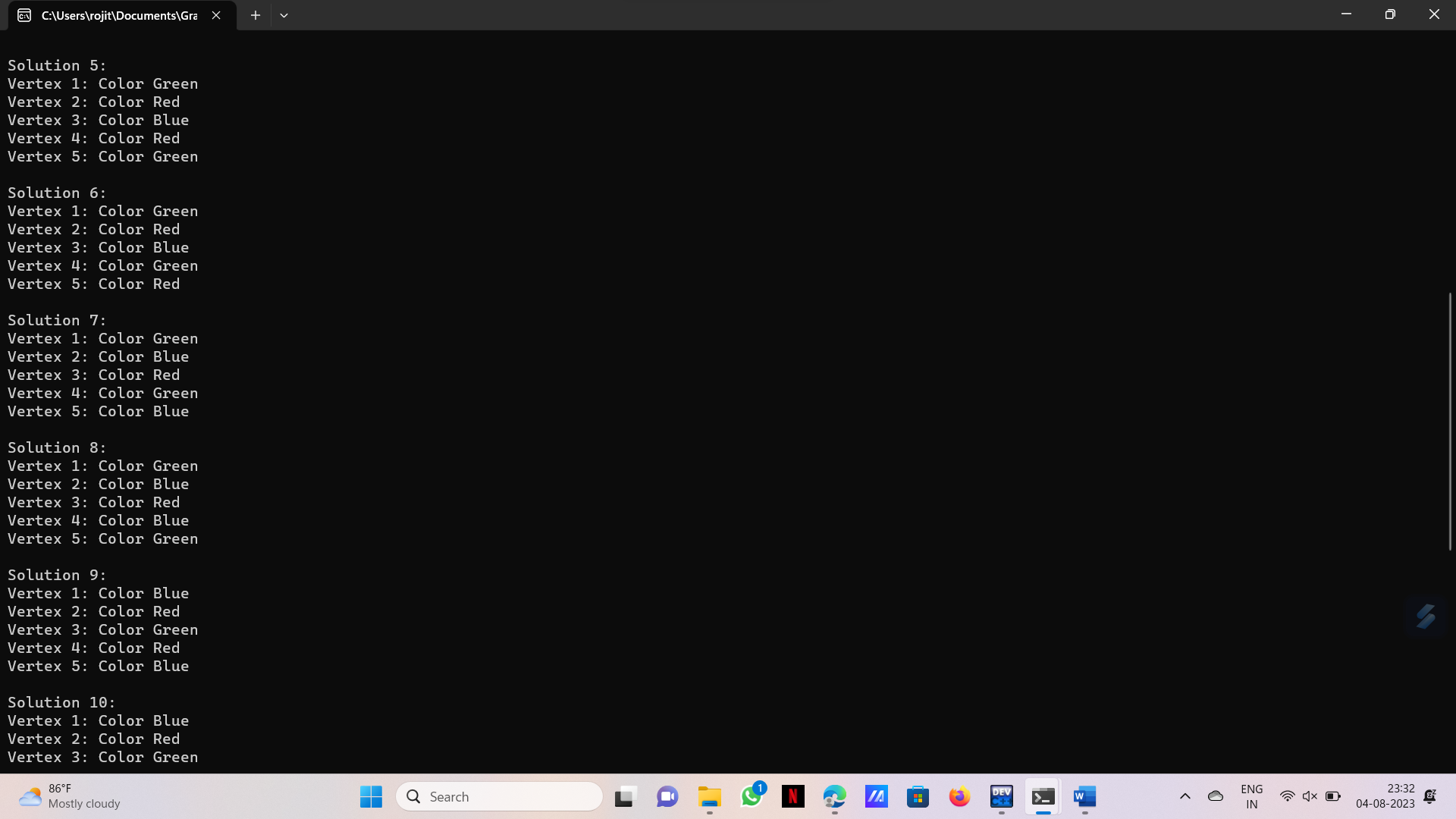
}

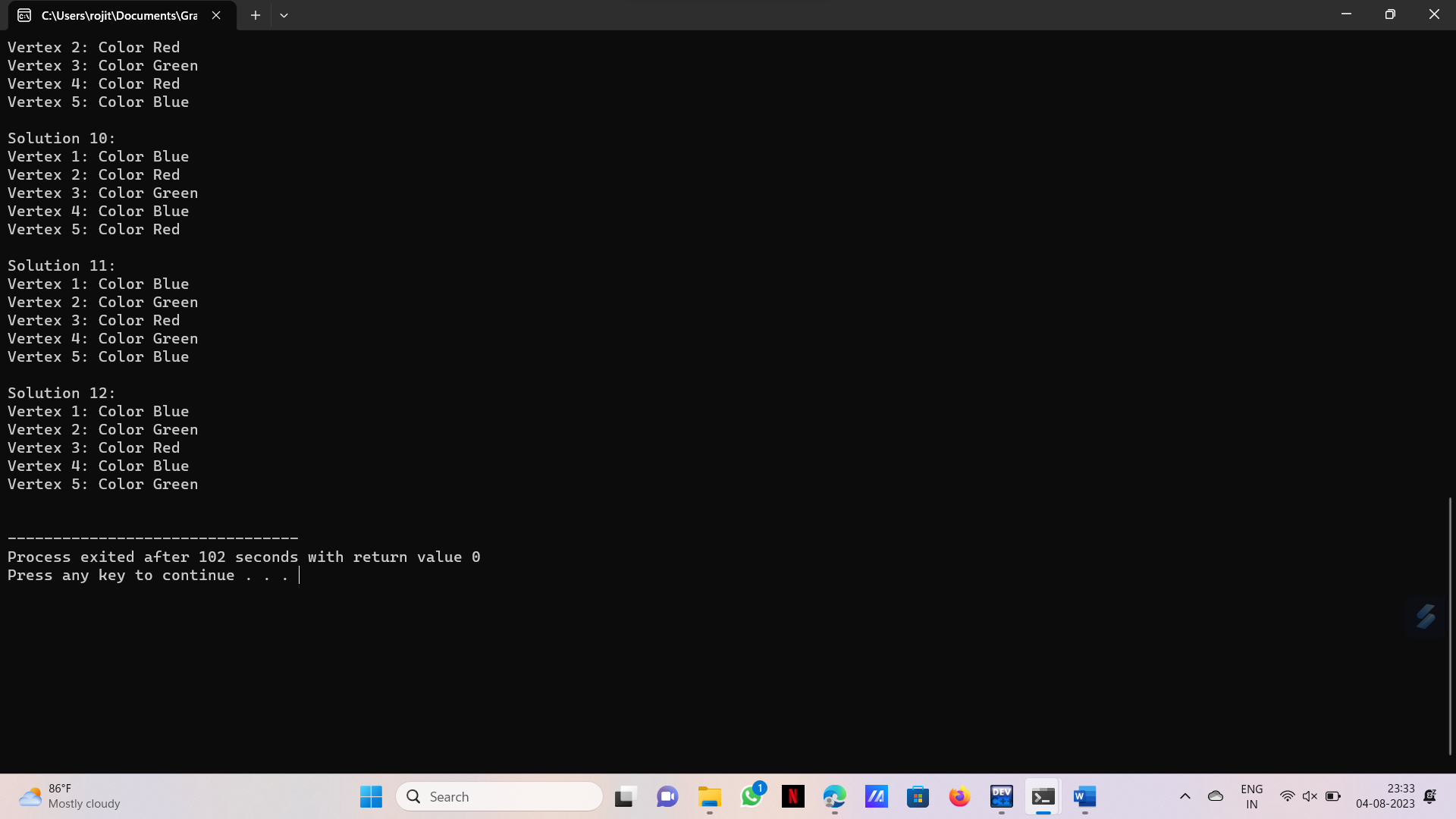
graphColoring();

}

Output:







4.Write a program to compute container loader problem:

Program:

#include <stdio.h>

#include <stdbool.h>

#define MAX\_ITEMS 100

void findCombinations(int items[], int n, int capacity, int currIndex, int currSum, bool selected[], int selectedCount) {

if (currSum > capacity) {

return;

}

if (currSum == capacity) {

printf("Possible solution: (");

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

if (selected[i]) {

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

selectedCount--;

if (selectedCount > 0) {

printf(", ");

}

}

}

printf(")\n");

return;

}

if (currIndex == n) {

return;

}

selected[currIndex] = true;

findCombinations(items, n, capacity, currIndex + 1, currSum + items[currIndex], selected, selectedCount + 1);

selected[currIndex] = false;

findCombinations(items, n, capacity, currIndex + 1, currSum, selected, selectedCount);

}

int main() {

int n;

printf("Enter the number of items: ");

scanf("%d", &n);

int items[MAX\_ITEMS];

printf("Enter the item weights: ");

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

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

}

int capacity;

printf("Enter the container capacity: ");

scanf("%d", &capacity);

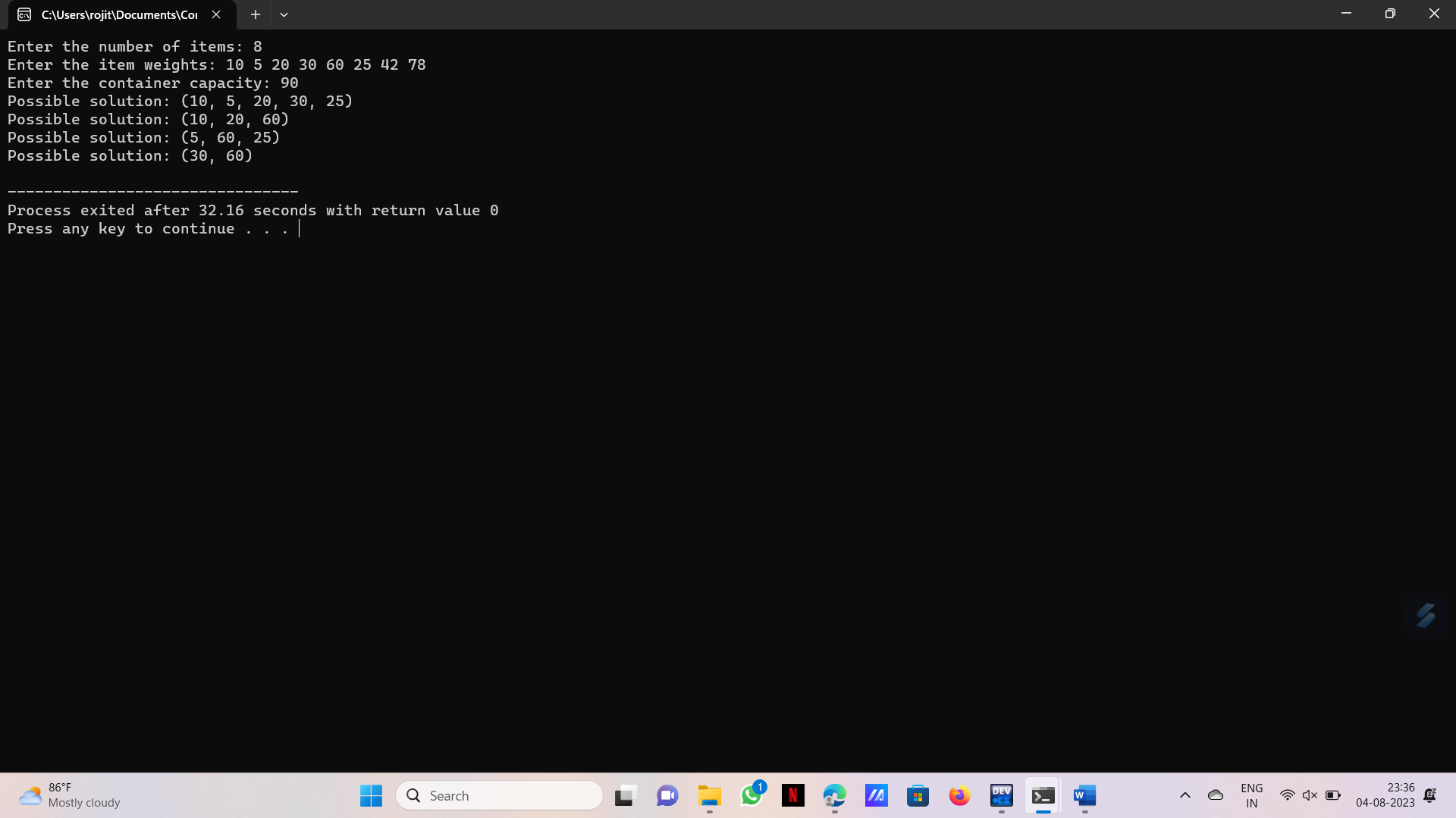
bool selected[MAX\_ITEMS] = {false};

findCombinations(items, n, capacity, 0, 0, selected, 0);

return 0;

}

Output:



5.Write a program to generate the list of all factors for a value:

Program:

#include <stdio.h>

void generateFactors(int n) {

printf("Factors of %d: ", n);

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

if (n % i == 0) {

printf("%d ", i);

}

}

printf("\n");

}

int main() {

int n;

printf("Enter a positive integer: ");

scanf("%d", &n);

if (n <= 0) {

printf("Invalid input. Please enter a positive integer.\n");

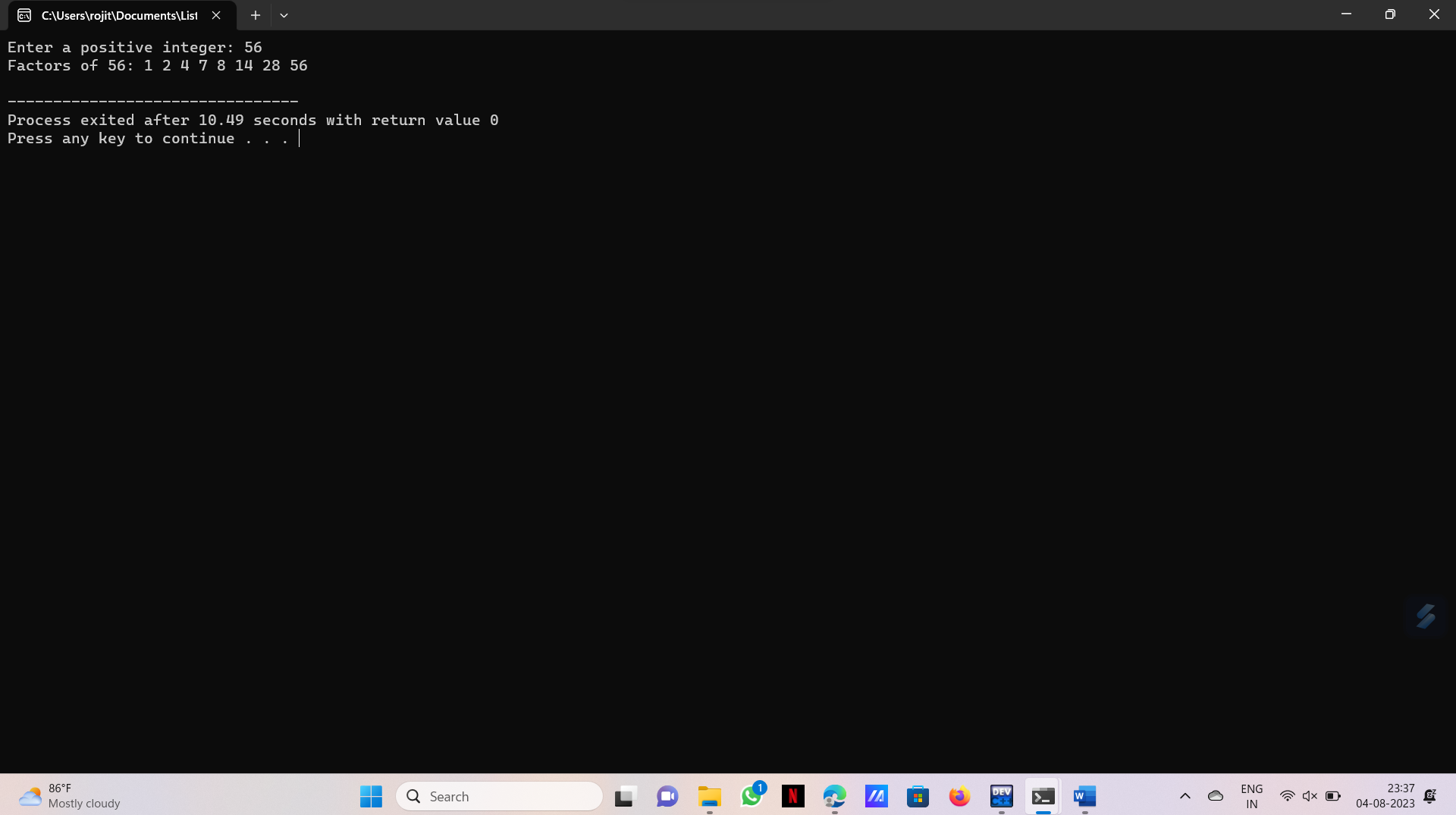
return 1;

}

generateFactors(n);

}

Output:



6.Write a program to perform linear search:

Program:

#include <stdio.h>

int linearSearch(int arr[], int size, int target, int \*comparisons) {

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

(\*comparisons)++;

if (arr[i] == target) {

return i;

}

}

return -1;

}

int main() {

int size;

printf("Enter the size of the array: ");

scanf("%d", &size);

int arr[size];

printf("Enter the elements of the array:\n");

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

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

}

int target;

printf("Enter the element to search for: ");

scanf("%d", &target);

int comparisons = 0;

int position = linearSearch(arr, size, target, &comparisons);

if (position != -1) {

printf("Element found at index: %d\n", position);

} else {

printf("Element not found in the array.\n");

}

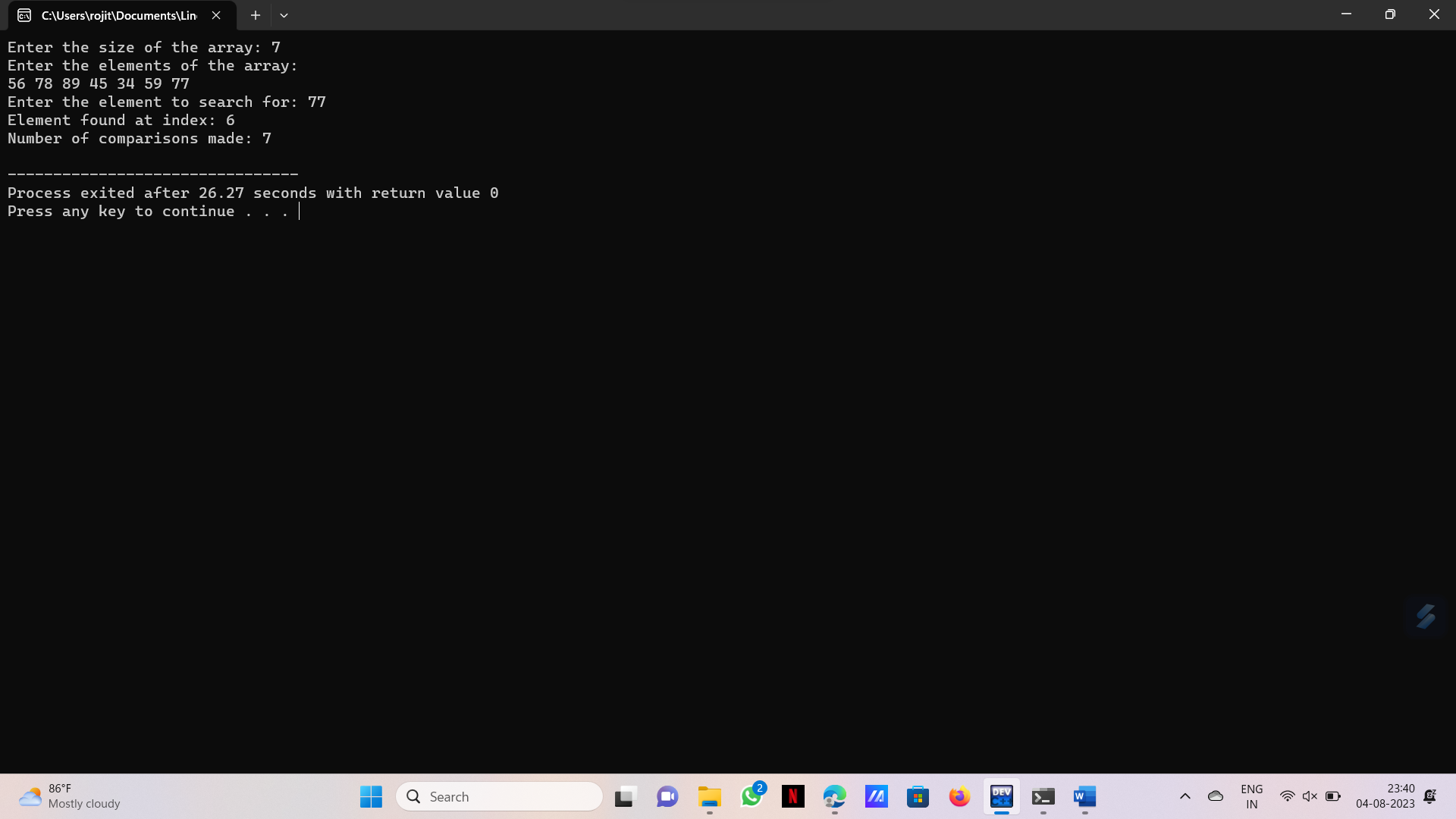
printf("Number of comparisons made: %d\n", comparisons);

return 0;

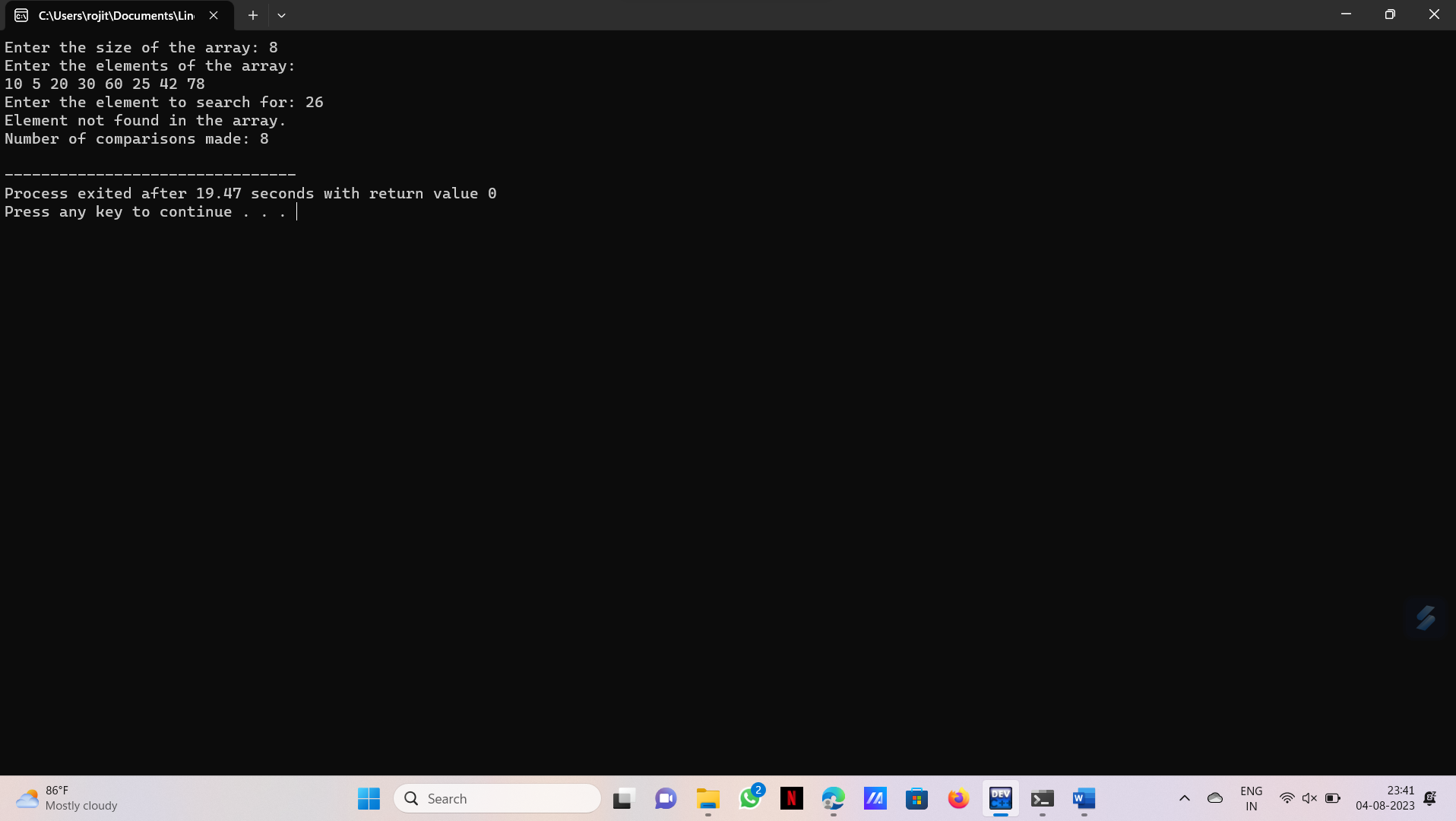
}

Output:

Successful search:



Unsuccessful search:



7.Write a program to find out Hamiltanian circuit using backtracking method:

Program:

#include <stdio.h>

#include <stdbool.h>

#define MAX\_VERTICES 10

int graph[MAX\_VERTICES][MAX\_VERTICES];

int path[MAX\_VERTICES];

bool visited[MAX\_VERTICES];

int numVertices;

void initialize() {

for (int i = 0; i < MAX\_VERTICES; i++) {

visited[i] = false;

path[i] = -1;

for (int j = 0; j < MAX\_VERTICES; j++) {

graph[i][j] = 0;

}

}

}

void readGraph() {

printf("Enter the number of vertices: ");

scanf("%d", &numVertices);

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

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

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

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

}

}

}

void printPath() {

printf("Hamiltonian Circuit: ");

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

printf("%c ", 'A' + path[i]);

}

printf("%c\n", 'A' + path[0]);

}

bool isSafe(int v, int pos) {

if (!graph[path[pos - 1]][v])

return false;

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

if (path[i] == v)

return false;

}

return true;

}

void findAllHamiltonians(int pos) {

if (pos == numVertices) {

if (graph[path[pos - 1]][path[0]]) {

printPath();

}

return;

}

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

if (isSafe(v, pos)) {

path[pos] = v;

findAllHamiltonians(pos + 1);

path[pos] = -1;

}

}

}

int main() {

initialize();

readGraph();

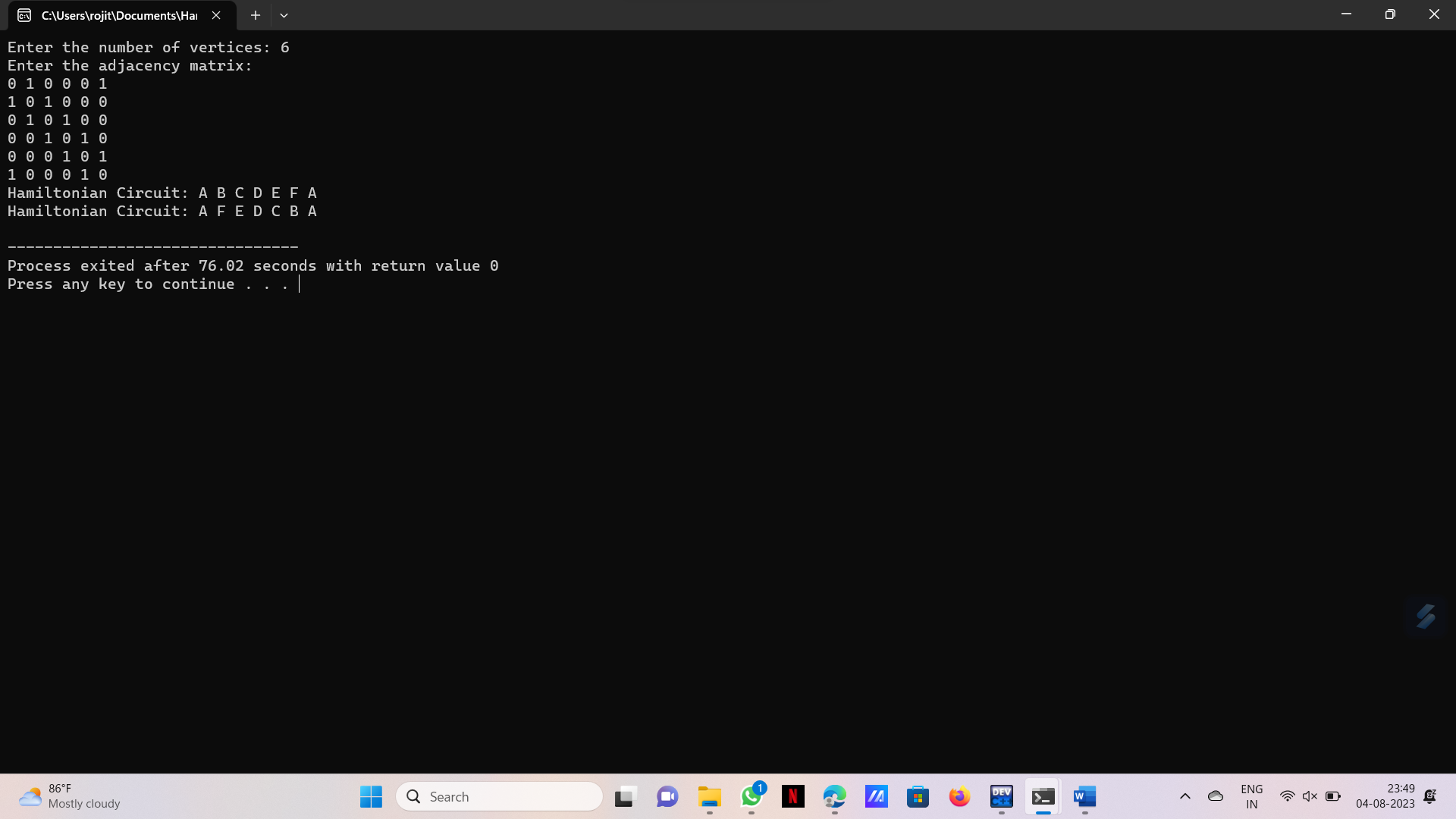
path[0] = 0;

findAllHamiltonians(1);

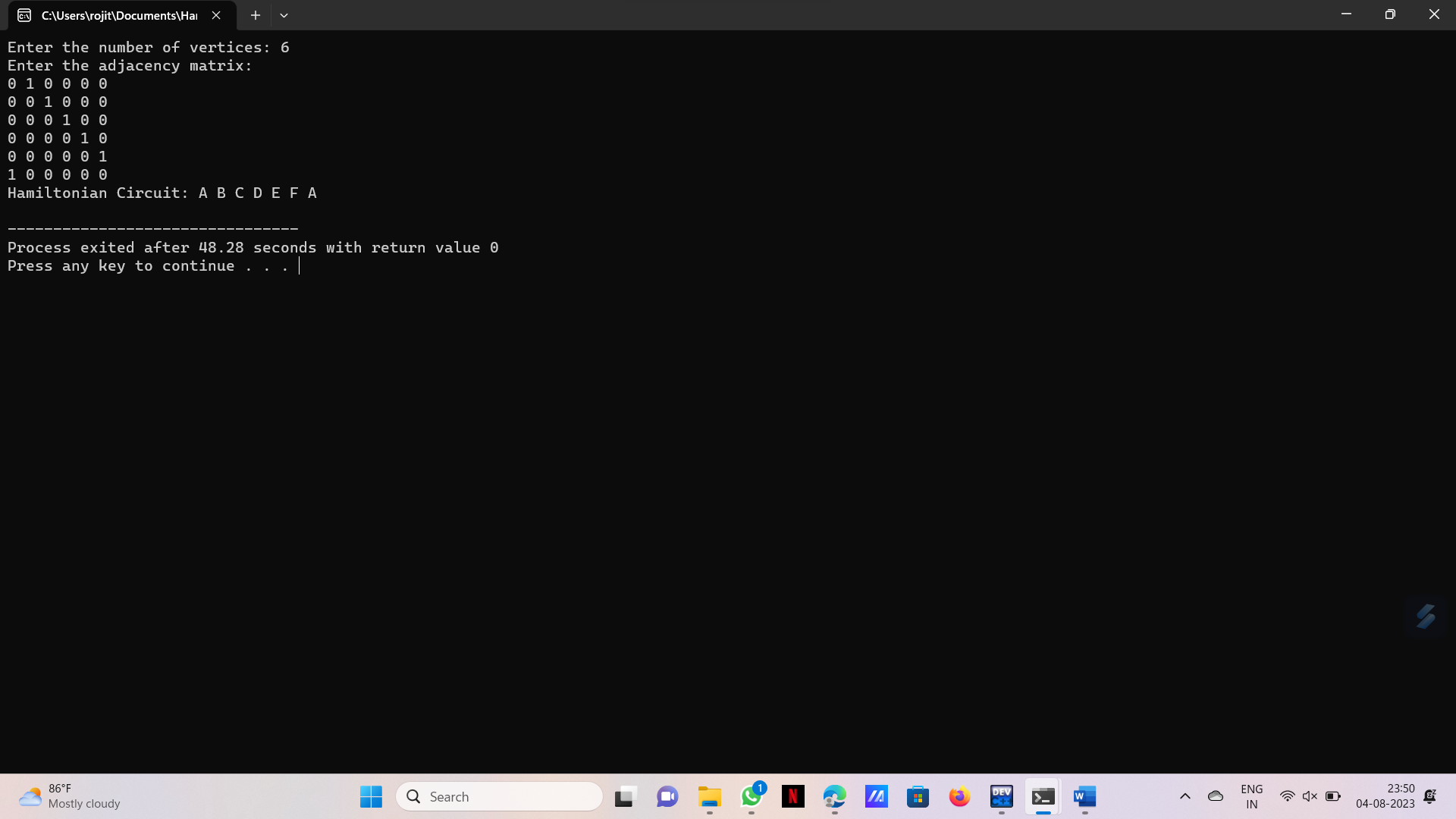
}

Output:

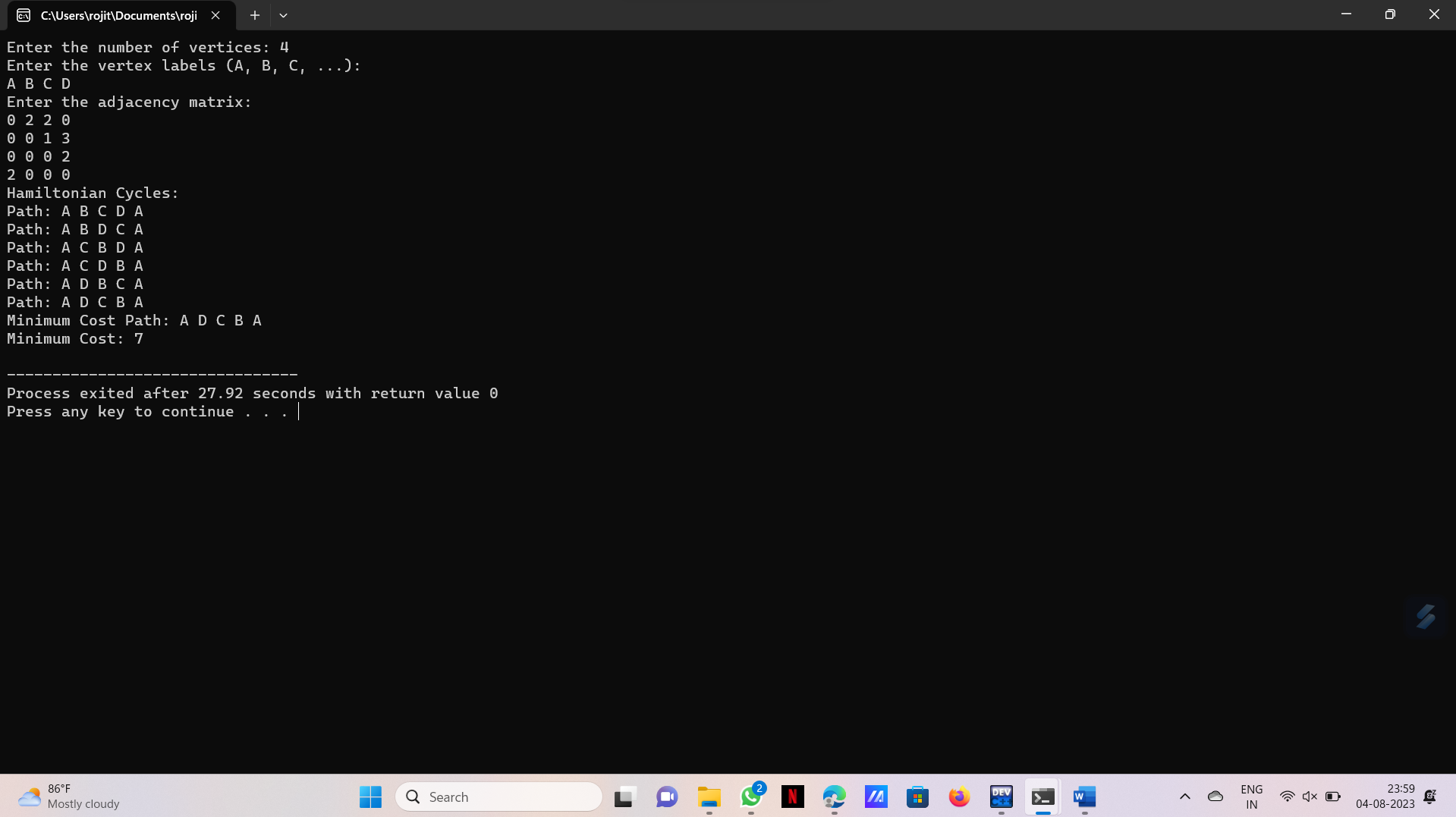
Undirected graph:



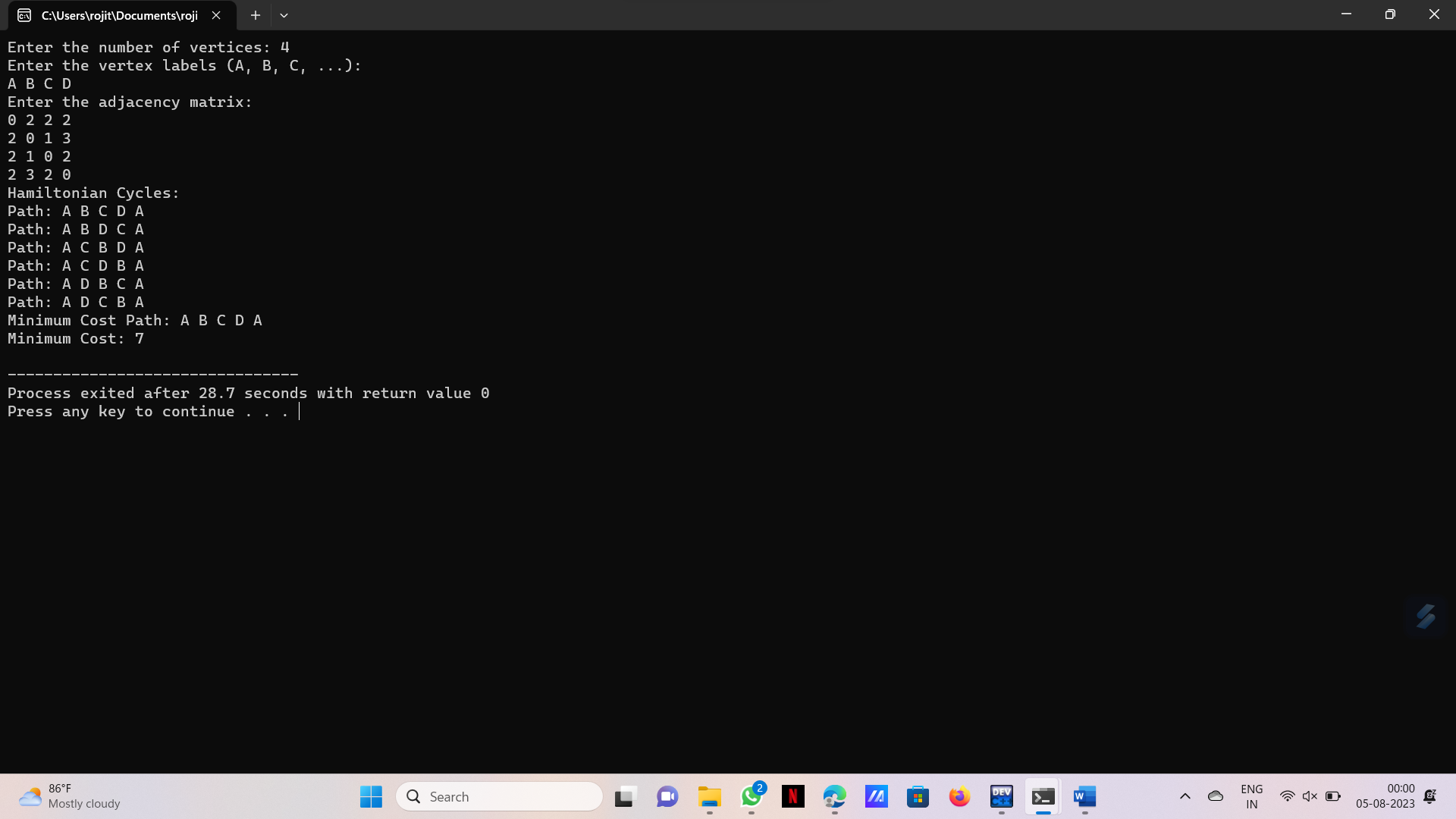
Directed graph:



Weighted Directed graph:



Weighted Undirected graph:



8.Write a program to perform Assignment problem using branch and bound:

Program:

#include <stdio.h>

#include <stdbool.h>

#include <limits.h>

#define MAX 10

int n;

int cost[MAX][MAX];

bool assigned[MAX];

int minCost = INT\_MAX;

void inputCostMatrix() {

printf("Enter the number of workers/tasks (maximum %d): ", MAX);

scanf("%d", &n);

printf("Enter the cost matrix (%dx%d):\n", n, n);

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

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

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

}

}

}

void branchAndBound(int worker, int currentCost) {

if (worker == n) {

if (currentCost < minCost) {

minCost = currentCost;

}

return;

}

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

if (!assigned[task]) {

assigned[task] = true;

branchAndBound(worker + 1, currentCost + cost[worker][task]);

assigned[task] = false;

}

}

}

int main() {

inputCostMatrix();

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

assigned[i] = false;

}

branchAndBound(0, 0);

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

}

Output:

