***1. Spiral Matris***

using System;

class SpiralMatrix

{

static void Main(string[] args)

{

// Giriş: NxN boyutunu kullanıcıdan al

Console.Write("Matris boyutunu girin (N): ");

int n = int.Parse(Console.ReadLine());

// Çıkış: NxN boyutunda bir matris oluştur

int[,] matrix = new int[n, n];

int value = 1; // Matriste kullanılacak değer

int top = 0, bottom = n - 1, left = 0, right = n - 1;

// Kontrol: Spiral şekilde matrisi doldur

while (value <= n \* n)

{

// Üst kenarı doldur

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

{

matrix[top, i] = value++;

}

top++;

// Sağ kenarı doldur

for (int i = top; i <= bottom; i++)

{

matrix[i, right] = value++;

}

right--;

// Alt kenarı doldur

for (int i = right; i >= left; i--)

{

matrix[bottom, i] = value++;

}

bottom--;

// Sol kenarı doldur

for (int i = bottom; i >= top; i--)

{

matrix[i, left] = value++;

}

left++;

}

// Matrisi ekrana yazdır

Console.WriteLine("Spiral Matris:");

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

{

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

{

Console.Write(matrix[i, j] + "\t");

}

Console.WriteLine();

}

}

}

***2. İki Matrisin Çarpımı***

using System;

class MatrixMultiplication

{

static void Main(string[] args)

{

// Giriş: Kullanıcıdan matris boyutunu al

Console.Write("Matris boyutunu girin (N): ");

int n = int.Parse(Console.ReadLine());

// İki NxN matris oluştur

int[,] matrixA = new int[n, n];

int[,] matrixB = new int[n, n];

int[,] resultMatrix = new int[n, n];

// Kullanıcıdan Matris A'yı al

Console.WriteLine("Matris A'yı girin:");

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

{

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

{

matrixA[i, j] = int.Parse(Console.ReadLine());

}

}

// Kullanıcıdan Matris B'yi al

Console.WriteLine("Matris B'yi girin:");

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

{

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

{

matrixB[i, j] = int.Parse(Console.ReadLine());

}

}

// Kontrol: Matrislerin çarpımını hesapla

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

{

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

{

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

{

resultMatrix[i, j] += matrixA[i, k] \* matrixB[k, j];

}

}

}

// Çıkış: Sonucu ekrana yazdır

Console.WriteLine("Matrislerin çarpımı:");

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

{

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

{

Console.Write(resultMatrix[i, j] + "\t");

}

Console.WriteLine();

}

}

}

***3. N'e Kadar Asal Sayıların Toplamı***

using System;

class PrimeSum

{

static void Main(string[] args)

{

// Giriş: Kullanıcıdan N sayısını al

Console.Write("N sayısını girin: ");

int n = int.Parse(Console.ReadLine());

int sum = 0; // Asal sayıların toplamı

// Kontrol: 2'den N'e kadar asal sayıları bul ve toplamını hesapla

for (int i = 2; i <= n; i++)

{

if (IsPrime(i))

{

sum += i; // Asal ise toplamı güncelle

}

}

// Çıkış: Toplamı ekrana yazdır

Console.WriteLine($"1'den {n}'e kadar olan asal sayıların toplamı: {sum}");

}

// Matematik adımı: Asal sayının kontrol fonksiyonu

static bool IsPrime(int number)

{

if (number < 2) return false; // 2'den küçük sayılar asal değildir

for (int i = 2; i <= Math.Sqrt(number); i++)

{

if (number % i == 0) return false; // Bölünebiliyorsa asal değildir

}

return true; // Asaldır

}

}

***4. Robotların Kurtarma Görevi***

using System;

using System.Collections.Generic;

class RobotRescue

{

static int[,] directions = new int[,]

{

{ -1, 0 }, // yukarı

{ 1, 0 }, // aşağı

{ 0, -1 }, // sola

{ 0, 1 } // sağa

};

static void Main(string[] args)

{

// Giriş: Grid boyutunu ve robot başlangıç pozisyonlarını al

Console.Write("Grid boyutunu girin (N): ");

int n = int.Parse(Console.ReadLine());

int[,] grid = new int[n, n];

// Grid'i kullanıcıdan al

Console.WriteLine("Grid'i girin (1 = sağlam, 0 = zarar görmüş):");

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

{

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

{

grid[i, j] = int.Parse(Console.ReadLine());

}

}

// Robot başlangıç pozisyonları

List<(int, int)> robotPositions = new List<(int, int)>();

Console.WriteLine("Robot başlangıç pozisyonlarını girin (x,y) formatında, 'q' ile çıkın:");

while (true)

{

string input = Console.ReadLine();

if (input.ToLower() == "q") break;

var parts = input.Split(',');

robotPositions.Add((int.Parse(parts[0]), int.Parse(parts[1])));

}

// Kontrol: Her robot için kurtarılan düğüm sayısını hesapla

HashSet<(int, int)> visited = new HashSet<(int, int)>();

int totalRescued = 0;

foreach (var pos in robotPositions)

{

totalRescued += Rescue(pos.Item1, pos.Item2, grid, visited);

}

// Çıkış: Toplam kurtarılan düğüm sayısını ekrana yazdır

Console.WriteLine($"Toplam kurtarılan düğüm sayısı: {totalRescued}");

}

// Matematik adımı: Robotların kurtarma işlevi

static int Rescue(int x, int y, int[,] grid, HashSet<(int, int)> visited)

{

if (x < 0 || x >= grid.GetLength(0) || y < 0 || y >= grid.GetLength(1) || grid[x, y] == 0 || visited.Contains((x, y)))

return 0; // Geçersiz pozisyon veya daha önce ziyaret edilmişse 0 döndür

visited.Add((x, y)); // Düğümü ziyaret et

int count = 1; // Kurtarılan düğüm sayısı

// Komşu düğümlere git

for (int i = 0; i < 4; i++)

{

count += Rescue(x + directions[i, 0], y + directions[i, 1], grid, visited);

}

return count; // Kurtarılan düğüm sayısını döndür

}

}

***5. Labirentte En Kısa Yol Bulma***

using System;

using System.Collections.Generic;

class MazeSolver

{

static int[,] directions = new int[,]

{

{ -1, 0 }, // yukarı

{ 1, 0 }, // aşağı

{ 0, -1 }, // sola

{ 0, 1 } // sağa

};

static void Main(string[] args)

{

// Giriş: Labirent boyutunu al

Console.Write("Labirent boyutunu girin (N): ");

int n = int.Parse(Console.ReadLine());

int[,] maze = new int[n, n];

// Kullanıcıdan labirenti al

Console.WriteLine("Labirenti girin (1 = yürünebilir, 0 = tuzak):");

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

{

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

{

maze[i, j] = int.Parse(Console.ReadLine());

}

}

// Çıkış: En kısa yolu bul

int steps = FindShortestPath(maze, n);

if (steps == -1)

{

Console.WriteLine("Yol Yok");

}

else

{

Console.WriteLine($"En Kısa Yol: {steps} adım");

}

}

// Matematik adımı: En kısa yolu bulma işlevi

static int FindShortestPath(int[,] maze, int n)

{

if (maze[0, 0] == 0 || maze[n - 1, n - 1] == 0) return -1; // Başlangıç veya bitiş geçersizse

Queue<(int, int, int)> queue = new Queue<(int, int, int)>();

queue.Enqueue((0, 0, 0)); // (x, y, adım sayısı)

bool[,] visited = new bool[n, n];

visited[0, 0] = true;

while (queue.Count > 0)

{

var (x, y, steps) = queue.Dequeue();

// Hedefe ulaştıysak adım sayısını döndür

if (x == n - 1 && y == n - 1) return steps;

// Komşu düğümlere git

for (int i = 0; i < 4; i++)

{

int newX = x + directions[i, 0];

int newY = y + directions[i, 1];

if (IsValid(newX, newY, n, maze, visited))

{

visited[newX, newY] = true; // Ziyaret et

queue.Enqueue((newX, newY, steps + 1)); // Bir sonraki pozisyona ekle

}

}

}

return -1; // Hedefe ulaşılamadıysa

}

// Kontrol: Geçerli hücreyi kontrol et

static bool IsValid(int x, int y, int n, int[,] maze, bool[,] visited)

{

return x >= 0 && x < n && y >= 0 && y < n && maze[x, y] == 1 && !visited[x, y];

}

}