**Лабораторная работа №5**

**Динамическое программирование**

**14 вариант**

*Составить программы, решающие следующие задачи. Входные данные находятся в текстовом файле input.txt. В первой строке файла указаны через пробел размеры массива, каждый из которых не превосходит 20. В последующих строках перечислены элементы массива - целые числа. Результатом работы программы должны быть оптимальное значение целевой функции и путь, при котором оно достигается.*

*Для указания пути (кроме задачи 6) вывести массив, отметив элементы пути либо другим цветом, либо некоторым другим образом.*

*2. Дан двумерный числовой массив размером N1xN1, где N1 - нечетное. Найти такой путь из центра массива к одной из сторон, чтобы сумма чисел по данному пути была минимальной. Из каждой клетки массива допустимо двигаться влево, вправо, вниз или вверх, при этом нельзя возвращаться в уже посещенные строки и столбцы.*

using System;

using System.Collections.Generic;

using System.Security.Cryptography;

class Program

{

public struct Point

{

public int X;

public int Y;

public Point(int x, int y)

{

X = x;

Y = y;

}

}

static int n, m;

static int[][] matrix;

static Dictionary<Point, Point> points;

static int[][] recalculatedMatrix;

static void Main()

{

ReadInput();

points = new Dictionary<Point, Point>();

recalculatedMatrix = new int[n][];

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

{

recalculatedMatrix[i] = new int[m];

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

{

recalculatedMatrix[i][j] = int.MaxValue;

}

}

int nowPointX = m / 2;

int nowPointY = n / 2;

recalculatedMatrix[nowPointY][nowPointX] = matrix[nowPointY][nowPointX];

points[new Point(nowPointY, nowPointX)] = new Point(nowPointY, nowPointX);//Ключ точки=значению точки, когда доходим до этой точки - выходим. Только для центральной точки

int layer = n / 2 - 1;//номер слоя(см. Рисунок)

while (layer >= 0)

{

FillingLine(layer, layer + 1, layer + 1);

FillingColumn(layer, layer +1, layer + 1);

FillingColumn(n - layer - 1, layer +1, n - layer - 2);

FillingLine(n - layer - 1, layer + 1, n - layer - 2);

//Если элемент сверху меньше элемента слева, то угол равен сумме элемента слева и значения угла

if (recalculatedMatrix[layer][layer + 1] < recalculatedMatrix[layer + 1][layer])

{

recalculatedMatrix[layer][layer] = recalculatedMatrix[layer][layer + 1] + matrix[layer][layer];

points[new Point(layer, layer)] = new Point(layer, layer + 1);

RecountColumnValues(layer, layer + 1, n - layer - 1);

}

else

{

recalculatedMatrix[layer][layer] = recalculatedMatrix[layer + 1][layer] + matrix[layer][layer];

points[new Point(layer, layer)] = new Point(layer + 1, layer);

RecountRowValues(layer, layer + 1, n - layer - 1);

}

//Если элемент снизу меньше элемента слева

if (recalculatedMatrix[n - layer - 1][layer + 1] < recalculatedMatrix[n - layer - 2][layer])

{

recalculatedMatrix[n - layer - 1][layer] = recalculatedMatrix[n - layer - 1][layer + 1] + matrix[n - layer - 1][layer];

points[new Point(n - layer - 1, layer)] = new Point(n - layer - 1, layer + 1);

RecountColumnValues(layer, n - layer - 2, layer);

}

else

{

recalculatedMatrix[n - layer - 1][layer] = recalculatedMatrix[n - layer - 2][layer] + matrix[n - layer - 1][layer];

points[new Point(n - layer - 1, layer)] = new Point(n - layer - 2, layer);

RecountRowValues(n - layer - 1, layer + 1, n - layer - 1);

}

//Если элемент снизу меньше элемента слева

if (recalculatedMatrix[n - layer - 1][n - layer - 2] < recalculatedMatrix[n - layer - 2][n - layer - 1])

{

recalculatedMatrix[n - layer - 1][n - layer - 1] = recalculatedMatrix[n - layer - 1][n - layer - 2] + matrix[n - layer - 1][n - layer - 1];

points[new Point(n - layer - 1, n - layer - 1)] = new Point(n - layer - 1, n - layer - 2);

RecountColumnValues(n - layer - 1, n - layer - 2, layer);

}

else

{

recalculatedMatrix[n - layer - 1][n - layer - 1] = recalculatedMatrix[n - layer - 2][n - layer - 1] + matrix[n - layer - 1][n - layer - 1];

points[new Point(n - layer - 1, n - layer - 1)] = new Point(n - layer - 2, n - layer - 1);

RecountRowValues(n - layer - 1, n - layer - 2, layer);

}

if (recalculatedMatrix[layer][n - layer - 2] < recalculatedMatrix[layer + 1][n - layer - 1])

{

recalculatedMatrix[layer][n - layer - 1] = recalculatedMatrix[layer][n - layer - 2] + matrix[layer][n - layer - 1];

points[new Point(layer, n - layer - 1)] = new Point(layer, n - layer - 2);

RecountColumnValues(n - layer - 1, layer + 1, n - layer - 1);

}

else

{

recalculatedMatrix[layer][n - layer - 1] = recalculatedMatrix[layer + 1][n - layer - 1] + matrix[layer][n - layer - 1];

points[new Point(layer, n - layer - 1)] = new Point(layer + 1, n - layer - 1);

RecountRowValues(layer, n - layer - 2, layer);

}

layer -= 1;

}

int c = points.Count;

int[] minWay = GetMinWay();

int curX = minWay[2];

int curY = minWay[1];

int sum = recalculatedMatrix[curX][curY];

List<Point> path = new List<Point>();

path.Add(new Point(curX, curY));

while (true)

{

if (points[new Point(curX, curY)].X == curX && points[new Point(curX, curY)].Y == curY)

{

break;

}

path.Add(points[new Point(curX, curY)]);

Point curPoint = points[new Point(curX, curY)];

curX = curPoint.X;

curY = curPoint.Y;

}

Console.WriteLine("Сумма минимального пути: "+sum);

path.Reverse();

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

{

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

{

if (path.Contains(new Point(i, j)))

{

Console.ForegroundColor = ConsoleColor.Red;

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

}

else

{

Console.ForegroundColor = ConsoleColor.White;

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

}

}

Console.WriteLine();

}

static void ReadInput()

{

string[] lines = File.ReadAllLines("input.txt");

string[] firstLine = lines[0].Split();

n = int.Parse(firstLine[0]);

m = int.Parse(firstLine[1]);

if(n!=m){

Console.WriteLine(‘Неправильный ввод массива’)

}

if (n >20|| m > 20)

{

Console.WriteLine("Введите массив поменьше");

}

if (n%2== 0 || m % 2 == 0)

{

Console.WriteLine("Введите нечетный размер массива");

}

matrix = new int[n][];

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

{

string[] row = lines[i + 1].Split();

matrix[i] = new int[m];

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

{

matrix[i][j] = int.Parse(row[j]);

}

}

}

static void FillingLine(int row/\*номер строки\*/, int layer/\*отступ от начала строки/номер слоя\*/, int previousLine/\*слой, который ниже\*/)

{

int nowPointX = 0;

int nowPointY = 0;

int minDistance = int.MaxValue;

//ищем минимальный элемент

for (int elemFromLine = layer; elemFromLine < m - layer; elemFromLine++)//чтобы не затрагивать углы слоя, начинаем со второго элемента и заканчиваем предпоследним(относительно отступа)

{

int recalculatedValue = recalculatedMatrix[previousLine][elemFromLine] + matrix[row][elemFromLine];//

if (recalculatedValue < minDistance)

{

minDistance = recalculatedValue;

nowPointX = elemFromLine;

nowPointY = row;

}

}

recalculatedMatrix[nowPointY][nowPointX] = minDistance;

points[new Point(nowPointY, nowPointX)] = new Point(previousLine, nowPointX);//в словарь заносим предыдущий элемент

//Заполняем все, кроме углов, начиная от индекса мин.элемента справа

for (int i = nowPointX + 1; i < n - layer; i++)

{

if (recalculatedMatrix[row][i - 1] < recalculatedMatrix[previousLine][i])

{

recalculatedMatrix[row][i] = recalculatedMatrix[row][i - 1] + matrix[row][i];

points[new Point(row, i)] = new Point(row, i - 1);

}

else

{

recalculatedMatrix[row][i] = recalculatedMatrix[previousLine][i] + matrix[row][i];

points[new Point(row, i)] = new Point(previousLine, i);

}

}

//Заполняем все, кроме углов, начиная от индекса мин.элемента слева

for (int i = nowPointX - 1; i >= layer; i--)

{

if (recalculatedMatrix[row][i + 1] < recalculatedMatrix[previousLine][i])

{

recalculatedMatrix[row][i] = recalculatedMatrix[row][i + 1] + matrix[row][i];

points[new Point(row, i)] = new Point(row, i + 1);

}

else

{

recalculatedMatrix[row][i] = recalculatedMatrix[previousLine][i] + matrix[row][i];

points[new Point(row, i)] = new Point(previousLine, i);

}

}

}

static void FillingColumn(int column, int layer, int previousColumn)//тот же принцип только сверху/снизу

{

int nowPointX = 0;

int nowPointY = 0;

int minDistance = int.MaxValue;

for (int elemFromColumn = layer; elemFromColumn < m - layer; elemFromColumn++)

{

int recalculatedValue = recalculatedMatrix[elemFromColumn][previousColumn] + matrix[elemFromColumn][column];

if (recalculatedValue < minDistance)

{

minDistance = recalculatedValue;

nowPointX = column;

nowPointY = elemFromColumn;

}

}

recalculatedMatrix[nowPointY][nowPointX] = minDistance;

points[new Point(nowPointY, nowPointX)] = new Point(nowPointY, previousColumn);

for (int i = nowPointY + 1; i < n - layer; i++)

{

if (recalculatedMatrix[i - 1][column] < recalculatedMatrix[i][previousColumn])

{

recalculatedMatrix[i][column] = recalculatedMatrix[i - 1][column] + matrix[i][column];

points[new Point(i, column)] = new Point(i - 1, column);

}

else

{

recalculatedMatrix[i][column] = recalculatedMatrix[i][previousColumn] + matrix[i][column];

points[new Point(i, column)] = new Point(i, previousColumn);

}

}

for (int i = nowPointY - 1; i >= layer; i--)

{

if (recalculatedMatrix[i + 1][column] < recalculatedMatrix[i][previousColumn])

{

recalculatedMatrix[i][column] = recalculatedMatrix[i + 1][column] + matrix[i][column];

points[new Point(i, column)] = new Point(i + 1, column);

}

else

{

recalculatedMatrix[i][column] = recalculatedMatrix[i][previousColumn] + matrix[i][column];

points[new Point(i, column)] = new Point(i, previousColumn);

}

}

}

static void RecountRowValues(int row, int startPoint, int endPoint)

{

int step = 1;

if (startPoint > endPoint)

{

step = -1;

int temp = startPoint;

startPoint = endPoint;

endPoint = temp;

}

for (int i = startPoint; i != endPoint + step; i += step)

{

if (recalculatedMatrix[row][i - step] + matrix[row][i] >= recalculatedMatrix[row][i])

{

break;

}

recalculatedMatrix[row][i] = recalculatedMatrix[row][i - step] + matrix[row][i];

points[new Point(row, i)] = new Point(row, i - step);

}

}

//Пересчитывают значения в столбце относительно угла

static void RecountColumnValues(int column, int startPoint, int endPoint)

{

int step = 1;

if (startPoint > endPoint)

{

Console.WriteLine("{0},{1}", startPoint, endPoint);

step = -1;

//int temp = startPoint;

//startPoint = endPoint;

//endPoint = temp;

}

for (int i = startPoint; i < endPoint + step; i += step)

{

if (recalculatedMatrix[i - step][column] + matrix[i][column] >= recalculatedMatrix[i][column])

{

break;

}

recalculatedMatrix[i][column] = recalculatedMatrix[i - step][column] + matrix[i][column];

points[new Point(i, column)] = new Point(i - step, column);

}

}

static int[] GetMinWay()

{

int[] Left = [int.MaxValue, 0, 0];

int[] Right = [int.MaxValue, 0, 0];

int[] Up = [int.MaxValue, 0, 0];

int[] Down = [int.MaxValue, 0, 0];

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

{

if (recalculatedMatrix[0][i] < Up[0])

{

Up = [recalculatedMatrix[0][i], i, 0];

}

if (recalculatedMatrix[i][0] < Left[0])

{

Left = [recalculatedMatrix[i][0], 0, i];

}

if (recalculatedMatrix[n - 1][i] < Down[0])

{

Down = [recalculatedMatrix[n - 1][i], i, n - 1];

}

if (recalculatedMatrix[i][n - 1] < Right[0])

{

Right = [recalculatedMatrix[i][n - 1], n - 1, i];

}

}

int[] minP = Left;

if (Right[0] < minP[0])

{

minP = Right;

}

if (Up[0] < minP[0])

{

minP = Up;

}

if (Down[0] < minP[0])

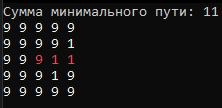
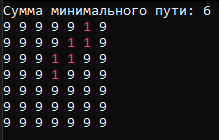
{

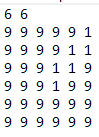
minP = Down;

}

return minP;

}

** **

**** ****

**Алгоритм работы:**

**Нулевой слой**

**первый слой**

* Заводим структуру Point, которая будет определять точку в матрице. Структура содержит два поля: X и Y, которые представляют координаты точки по оси Х и по оси Y соответственно. Далее определяется конструктор, который принимает два аргумента x и y и инициализирует поля значениями этих аргументов.
* Объявляем переменные n, m, matrix, points, recalculatedMatrix: n,m – размеры; начальная матрица; словарь, отвечающий за хранение переменных, включенных в минимальный путь; пересчитанная матрица уже с минимальным путем соответственно
* В функции main переходим к функции ReadInput, которая считывает сначала все строчки, в первой строке берем размеры массива, затем заполняем начальную матрицу элементами из матрицы из входного файла.
* Затем заполняем пересчитанную матрицу элементами с максимальным значением, чтобы было проще находить минимальный элемент
* Затем находим центр массива, присваиваем значение центрального элемента начальной матрицы центральному элементу пересчитанной матрицы, ведь мы начинаем искать минимальный путь из центра. В словарь мы также заносим значение центрального элемента. Вводим переменную, отвечающую за номер слоя.
* В цикле пока мы не дойдем до краев матрицы, мы начинаем искать минимальный путь не затрагивая углы матрицы в функции FillingLine(int row/\*номер строки\*/, int layer/\*отступ от начала строки/номер слоя\*/, int previousLine/\*слой, который ниже\*/) и FillingColumn(int column, int layer, int previousColumn). Сначала смотрим верхнюю строчку слоя, затем левый и правый столбец слоя, и нижнюю строчку слоя. В функции FillingLine вводим переменные для координат, и переменную для поиска минимальной суммы, пока что она равна максимальному значению. В цикле прибавляем значение элемента нового слоя к элементу предыдущего слоя и сравниваем, если значение суммы меньше максимального значения, то теперь минимальное значение равно этой сумме, а координаты равны координате элемента, где хранится сумма. Вносим в пересчитанную матрицу значение элемента суммы, а в словарь заносим предыдущий элемент до суммы. По такому же принципу проверяем элементы справа и слева. Также по такому же принципу работает функция FillingColumn.
* Затем начинаем заполнять углы слоя, проверяем значения справа и снизу от левого верхнего угла. К значению углового элемента прибавляем значение минимального из двух элементов, в словарь записываем минимальный из двух элементов. Вызываем функцию RecountColumnValues, которая пересчитывает значения столбца слоя относительно угла. Значение элемента ниже углового элемента равно сумме значения углового элемента и значения углового элемента из начальной матрицы. В словарь заносим предыдущий элемент. Также поступаем и с другими углами, но в некоторых случаях вызываем RecountRowValues, но он такой же по принципу, как и RecountColumnValues. В конце цикла уменьшаем уровень слоя.
* В функции GetMinWay() ищем минимальным путь. Создаем 4 массива для каждой из сторон в котором по 3 элемента, значение элемента и его координаты, и начинаем перебор, если элемент пересчитанной матрицы меньше значения элемента массива стороны, то он попадает в массив. Затем ищем наименьший из путей, создаем новый массив делаем его равным массиву Left, сравниваем с тремя другими массивами и выводим наименьший.
* Создаем переменные для текущих координат переменной, а также переменную суммы минимального пути, которая равна последнему элементу массива minWay. Создаем список и добавляем в него начальную точку. Создаем бесконечный цикл для поиска пути и проверяем условие завершения цикла, если значение координат точки из словаря равна текущей точке, то цикл прерывается. Добавляем следующие точки из словаря в path, создаем переменную в которую помещаем текущую точку из словаря, а в переменные координат записываем координаты точки.
* Выводим сумму, переворачиваем список для правильного вывода элементов и выводим матрицу с обновленными данными: элементы минимального пути мы помечаем красным цветом, остальные белым