Вариант 16

Дана целочисленная прямоугольная матрица. Определить:

количество отрицательных элементов в тех строках, которые содержат хотя бы один нулевой элемент, вывести номер каждой строки и количество отрицательных элементов в ней;

номера строк и столбцов всех седловых точек матрицы;

поменять местами строку и столбец , находящиеся на пересечении минимальной седловой точки, если точка находится на главной диагонали.

ПРИМЕЧАНИЕ

Матрица А имеет седловую точку Aij, если Aij является минимальным элементом в i-й строке и максимальным в j-м столбце.

Инициализация многомерных массивов

Значения элементов многомерного массива, как и в одномерном случае, могут быть заданы константными значениями при объявлении, заключенными в фигурные скобки {}. Однако в этом случае указание количества элементов в строках и столбцах должно быть обязательно указано в квадратных скобках [ ].

Пример

#include <stdio.h>

int main () {

int a[2][3]={1, 2, 3, 4, 5, 6};

printf ("%d %d %d\n", a[0][0], a[0][1], a[0][2]);

printf ("%d %d %d\n", a[1][0], a[1][1], a[1][2]);

return 0;

}

Результат выполнения

При работе с массива часто используется параметрический цикл, тот же результат выполнения программы будет достигнут следующим программным кодом:

for (i=0; i<2; i++) // цикл по строкам

{

for (j=0; j<3; j++) // цикл по столбцам

printf ("%d ",a[ i][j]);

printf (“\n”);

}

Массив а[2][3] можно инициализировать, явно выделяя строки массива:

int a[2][3]={{1, 2, 3}, {4, 5, 6}};

Можно инициализировать не все элементы массива, в нижеследующем инициализируются только элементы первого столбца:

int a[2][3]={{1}, {4}};

Следующее описание формирует «треугольное» заполнение матрицы в целочисленном массиве из 5 строк и 4 столбцов:

int x[5][4]={{1}, {2, 3},{4, 5, 6},{7,8,9,10}};

Однако чаще требуется вводить значения элементов многомерного массива в процессе выполнения программы. С этой целью удобно использовать вложенный параметрический цикл.

Пример

#include <stdio.h>

int main() {

int a[2][3]; // массив из 2 строк и 3 столбцов

int i, j;

// Ввод элементов массива

for(i=0; i<2; i++) // цикл по строкам

{

for(j=0; j<3; j++) // цикл по столбцам

{

printf("a[%d][%d] = ", i,j);

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

}

}

// Вывод элементов массива

for(i=0; i<2; i++) // цикл по строкам

{

for(j=0; j<3; j++) // цикл по столбцам

{

printf("%d ", a[i][j]);

}

printf("\n"); // перевод на новую строку

}

return 0;

}

Результат выполнения

ВЫПОЛНЕНО

#include <stdio.h>

#define MAX\_SIZE 100

*void* inputMatrix(*int* *matrix*[MAX\_SIZE][MAX\_SIZE], *int* \**rows*, *int* \**cols*) {

printf("Enter number of lines: ");

scanf("%d", *rows*);

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

scanf("%d", *cols*);

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

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

for (*int* j = 0; j < \**cols*; j++) {

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

}

}

}

*void* countNegativeAndZero(*int* *matrix*[MAX\_SIZE][MAX\_SIZE], *int* *rows*, *int* *cols*) {

printf("\nLine numbers and number of negative elements:\n");

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

*int* count\_negative = 0;

*int* has\_zero = 0;

for (*int* j = 0; j < *cols*; j++) {

if (*matrix*[i][j] < 0) {

count\_negative++;

}

if (*matrix*[i][j] == 0) {

has\_zero = 1;

}

}

if (has\_zero) {

printf("Line %d: %d negative elements\n", i + 1, count\_negative);

}

}

}

*void* findSaddlePoints(*int* *matrix*[MAX\_SIZE][MAX\_SIZE], *int* *rows*, *int* *cols*) {

printf("\nRow and column numbers of saddle points:\n");

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

for (*int* j = 0; j < *cols*; j++) {

*int* is\_min\_in\_row = 1;

for (*int* k = 0; k < *cols*; k++) {

if (*matrix*[i][k] < *matrix*[i][j]) {

is\_min\_in\_row = 0;

break;

}

}

*int* is\_max\_in\_col = 1;

for (*int* k = 0; k < *rows*; k++) {

if (*matrix*[k][j] > *matrix*[i][j]) {

is\_max\_in\_col = 0;

break;

}

}

if (is\_min\_in\_row && is\_max\_in\_col) {

printf("Saddle point: row %d, column%d\n", i + 1, j + 1);

}

}

}

}

*void* swapMinSaddlePointRowCol(*int* *matrix*[MAX\_SIZE][MAX\_SIZE], *int* *rows*, *int* *cols*) {

*int* min\_row = -1, min\_col = -1;

*int* min\_value = 2147483647; // Максимальное значение целого числа

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

for (*int* j = 0; j < *cols*; j++) {

*int* is\_min\_in\_row = 1;

for (*int* k = 0; k < *cols*; k++) {

if (*matrix*[i][k] < *matrix*[i][j]) {

is\_min\_in\_row = 0;

break;

}

}

*int* is\_max\_in\_col = 1;

for (*int* k = 0; k < *rows*; k++) {

if (*matrix*[k][j] > *matrix*[i][j]) {

is\_max\_in\_col = 0;

break;

}

}

if (is\_min\_in\_row && is\_max\_in\_col && *matrix*[i][j] < min\_value) {

min\_value = *matrix*[i][j];

min\_row = i;

min\_col = j;

}

}

}

if (min\_row != -1) {

printf("\nMinimum saddle point: row %d, column %d \n", min\_row + 1, min\_col + 1);

// Меняем местами строки и столбцы

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

// Сохраняем значения

*int* temp = *matrix*[i][min\_col];

*matrix*[i][min\_col] = *matrix*[min\_row][i];

*matrix*[min\_row][i] = temp;

}

printf("After exchanging row %d and column %d:\n", min\_row + 1, min\_col + 1);

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

for (*int* j = 0; j < *cols*; j++) {

printf("%d ", *matrix*[i][j]);

}

printf("\n");

}

} else {

printf("No saddle points found!\n");

}

}

*int* main() {

*int* matrix[MAX\_SIZE][MAX\_SIZE];

*int* rows, cols;

inputMatrix(matrix, &rows, &cols);

countNegativeAndZero(matrix, rows, cols);

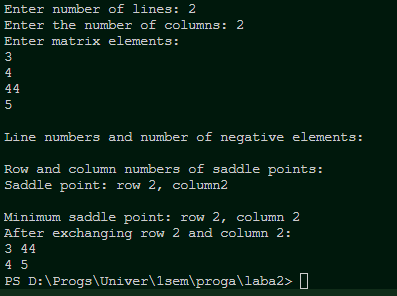
findSaddlePoints(matrix, rows, cols);

swapMinSaddlePointRowCol(matrix, rows, cols);

return 0;

}

результат



Разбор кода

Заголовочные файлы

#include <stdio.h>

* Подключает библиотеку стандартного ввода-вывода, что позволяет использовать такие функции, как printf для вывода и scanf для ввода данных.

#include <stdlib.h>

* Подключает библиотеку, содержащую функции для работы с памятью, преобразования типов и генерации случайных чисел с помощью rand().

#include <time.h>

* Подключает библиотеку для работы со временем, необходимую для инициализации генератора случайных чисел.

Определение максимального размера

#define MAX\_SIZE 100

* Определяет макрос MAX\_SIZE, который устанавливает максимальный размер матрицы (100 на 100).

Функция для вывода матрицы

void print\_matrix(float matrix[MAX\_SIZE][MAX\_SIZE], int n, int m) {

* Определяет функцию print\_matrix, принимающую в качестве параметров двумерный массив matrix и его размерности n (количество строк) и m (количество столбцов).

printf("Matrix:\n");

* Выводит заголовок "Matrix:" перед выводом содержимого матрицы.

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

* Запускает цикл, который проходит по всем строкам матрицы от 0 до n-1.

c

Copy

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

* Вложенный цикл, который проходит по всем столбцам текущей строки от 0 до m-1.

c

Copy

printf("%.2f ", matrix[i][j]);

* Выводит элемент матрицы с двумя знаками после запятой, добавляя пробел после каждого элемента.

c

Copy

printf("\n");

* После вывода всех элементов строки добавляет перевод строки.

c

Copy

printf("\n");

}

* Добавляет пустую строку для лучшей читаемости. Закрывает функцию.

Функция сортировки строки методом вставки

c

Copy

void insertion\_sort\_row(float matrix[MAX\_SIZE][MAX\_SIZE], int row, int m) {

* Определяет функцию insertion\_sort\_row, сортирующую строку с индексом row в матрице с m элементами.

c

Copy

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

* Запускает цикл, начиная с 1, поскольку первый элемент уже считается отсортированным.

c

Copy

float key = matrix[row][i];

* Сохраняет текущее значение для вставки в переменную key.

c

Copy

int j = i - 1;

* Устанавливает j на следующий элемент (левый от key).

c

Copy

while (j >= 0 && matrix[row][j] > key) {

* Запускает цикл, который продолжается, пока j не станет менее 0 и текущий элемент больше key.

c

Copy

matrix[row][j + 1] = matrix[row][j];

* Сдвигает текущий элемент на одну позицию вправо, чтобы освободить место для вставки key.

j--;

* Уменьшает значение j для перехода к следующему элементу для сравнения.

}

matrix[row][j + 1] = key;

* Вставляет key в освободившуюся позицию.

}

* Закрывает функцию.

Функция сортировки всех строк

void sort\_rows(float matrix[MAX\_SIZE][MAX\_SIZE], int n, int m) {

* Определяет функцию sort\_rows, сортирующую все строки в матрице.

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

* Запускает цикл по всем строкам от 0 до n-1.

insertion\_sort\_row(matrix, i, m);

* Вызывает функцию insertion\_sort\_row для сортировки строки i.

printf("Matrix after row sorting %d:\n", i + 1);

* Выводит сообщение об успешной сортировке строки.

print\_matrix(matrix, n, m);

* Вызывает функцию print\_matrix, чтобы вывести матрицу после сортировки текущей строки.

}

* Закрывает функцию.

Функция обмена столбцов

void swap\_columns(float matrix[MAX\_SIZE][MAX\_SIZE], int n, int col1, int col2) {

* Определяет функцию swap\_columns, которая обменивает значения в двух столбцах col1 и col2.

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

* Запускает цикл по всем строкам от 0 до n-1.

float temp = matrix[i][col1];

* Сохраняет значение из столбца col1 в временной переменной temp.

matrix[i][col1] = matrix[i][col2];

* Переносит значение из столбца col2 в col1.

matrix[i][col2] = temp;

* Устанавливает значение из temp во второй столбец col2.

}

* Закрывает функцию.

Функция сортировки столбцов

void sort\_columns(float matrix[MAX\_SIZE][MAX\_SIZE], int n, int m) {

* Определяет функцию sort\_columns, которая сортирует столбцы матрицы.

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

* Запускает цикл по всем столбцам от 0 до m-1.

for (int k = j + 1; k < m; k++) {

* Запускает вложенный цикл для сравнения текущего столбца с каждым последующим.

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

* Запускает цикл по всем строкам для каждого столбца.

if (matrix[i][j] > matrix[i][k]) {

* Если значение в столбце j больше значения в столбце k в одной строке...

swap\_columns(matrix, n, j, k);

* ...то обменивает столбцы j и k с помощью функции swap\_columns.

break; *// Сортируем только один раз для каждого сравнения*

* Прерывает внутренний цикл, чтобы избежать дальнейших сравнений для пары столбцов.

}

* Закрывает вложенный цикл.

printf("Matrix after column sorting %d:\n", j + 1);

* Выводит сообщение о завершении сортировки текущего столбца.

print\_matrix(matrix, n, m);

* Вызывает print\_matrix для отображения матрицы после сортировки текущего столбца.

}

* Закрывает функцию.

Основная функция

int main() {

* Начинает основную часть программы с функции main.

int n, m;

* Объявляет переменные n и m для хранения количества строк и столбцов.

float matrix[MAX\_SIZE][MAX\_SIZE];

* Объявляет двумерный массив matrix, который будет использоваться для хранения значений матрицы.

printf("Enter the number of rows (n) and columns (m): ");

* Выводит текст для приглашения пользователя ввести количество строк и столбцов.

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

* Читает введенные значения и сохраняет их в переменные n и m.

srand(time(0));

* Инициализирует генератор случайных чисел с использованием текущего времени, чтобы каждое выполнение программы получало разные случайные числа.

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

* Начинает цикл по строкам от 0 до n-1.

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

* Вложенный цикл по столбцам от 0 до m-1.

matrix[i][j] = (float)(rand() % 100) / 10.0; *// Случайные числа от 0 до 9.9*

* Генерирует случайное число от 0 до 99, делит его на 10, чтобы получить дробное число от 0 до 9.9 и сохраняет его в матрице.

}

}

* Закрывает циклы для заполнения матрицы.

printf("Original matrix:\n");

* Выводит сообщение о том, что будет представлена исходная матрица.

print\_matrix(matrix, n, m);

* Вызывает функцию print\_matrix, чтобы показать исходную матрицу.

sort\_rows(matrix, n, m);

* Вызывает функцию sort\_rows, чтобы отсортировать все строки матрицы.

sort\_columns(matrix, n, m);

* Вызывает функцию sort\_columns, чтобы отсортировать все столбцы матрицы.

return 0;

}

* Завершает выполнение программы, возвращая 0, что сигнализирует о нормальном завершении.