МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра прикладной математики

Разреженные матрицы

ОТЧЕТ

По лабораторной работе

по дисциплине

Технология программирования

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Санников А.Н.

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Зырянов Е.А.

22-ПМ-1

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Задание на лабораторную работу:

1. Дана разреженная матрица (**Диагональный формат**). Найти сумму её элементов aij , у которых сумма (i+j) является чётной.

Цель работы:

Познакомиться с разреженными матрицами и форматами их хранения.

Ход работы:

1) Написали программу:

#include <iostream>

#include <fstream>

struct node { //структура элемента

private: //

int data; //

node\* next; //

public: //

node() { //

next = NULL; //

} //

int get\_data() { //

return data; //

} //

node\* get\_next() { //

return next; //

} //

void set\_data(int data) { //

this->data = data; //

} //

void set\_next(node\* next) { //

this->next = next; //

} //

int\* get\_data\_link() { //

return &data; //

} //

};

class list { //список

private: //

node\* head = NULL;

public:

void push\_back(int data) { //вставить в конец

if (head == NULL) { //

head = new node; //

head->set\_data(data); //

return; //

} //

node\* tmp = head; //

while (tmp->get\_next()) { //

tmp = tmp->get\_next(); //

} //

tmp->set\_next(new node); //

tmp->get\_next()->set\_data(data); //

}

void clear() {

node\* current = head;

while (current) {

node\* next = current->get\_next();

delete current;

current = next;

}

head = NULL;

}

void print() { //вывести

node\* tmp = head; //

while (tmp) { //

std::cout << tmp->get\_data() << ' '; //

tmp = tmp->get\_next(); //

} //

std::cout << '\n'; //

}

int len() {

node\* tmp = head;

int c = 0;

while (tmp) { //

c += 1; //

tmp = tmp->get\_next(); //

}

return c;

}

int\* operator[](int i) { //перегрузка оператора

node\* tmp = head; //

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

tmp = tmp->get\_next(); //

} //

return tmp->get\_data\_link(); //

}

void remove\_duplicates() {

if (head == NULL) {

return; // Список пуст, ничего удалять не нужно

}

node\* current = head;

while (current) {

node\* runner = current;

while (runner->get\_next()) {

if (\*(runner->get\_next()->get\_data\_link()) == current->get\_data()) {

// Удаляем повторяющийся элемент

node\* duplicate = runner->get\_next();

runner->set\_next(duplicate->get\_next());

delete duplicate;

}

else {

runner = runner->get\_next();

}

}

current = current->get\_next();

}

}

void values\_to\_matrix(int\*\* packed\_matrix, int digs, int elems) {

node\* tmp = head;

int i = 0, j = 0;

while (tmp) {

while ((tmp->get\_data()) != -999) {

packed\_matrix[i][j] = tmp->get\_data();

++j;

tmp = tmp->get\_next();

}

++i; j = 0;

tmp = tmp->get\_next();

}

}

};

int chooseOption()

{

int option; // Функция выбора опции

std::cout << "Как вы хотите ввести матрицу? " << std::endl; //

std::cout << "1 - из консоли. 2 - из файла.: " << std::endl; //

std::cin >> option; //

if (option == 1) //

return 1; //

else if (option == 2) //

return 2; //

else { //

std::cout << "Неверное значение." << std::endl; //

return -1; //

} //

}

void readFromConsole(int\*\* Matrix, int rows, int cols) {

std::cout << "Введите матрицу:" << std::endl; // Функция считывания матрицы из консоли.

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

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

std::cin >> Matrix[i][j]; //

} //

} //

}

void readFromFile(int\*\* Matrix, int rows, int cols) {

std::string inputFileName = ""; // Функция считывания матрицы из консоли.

std::cout << "Введите название файла с матрицей: "; //

std::cin >> inputFileName; // //

std::ifstream in(inputFileName); //

if (in.is\_open()) { //

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

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

in >> Matrix[i][j]; //

} //

} //

} //

else { //

std::cout << "Файл не найден. "; //

readFromFile(Matrix, rows, cols); //

} //

in.close(); //

}

int abs(int n) {

if (n >= 0) {

return n;

}

else {

return -n;

}

}

void printMatrix(int\*\* matrix, const int rows, const int cols) {

for (int i = 0; i < rows; ++i) { // функция выведения матрицы в консоль.

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

std::cout << matrix[i][j] << " "; //

} //

std::cout << std::endl; //

}

}

int\*\* unpackMatrix(int\*\* packedMatrix, const int\* index, const int rows, const int cols) {

const int size = cols; // функция распаковки исходной матрицы.

int\*\* unpackedMatrix = new int\* [size]; //

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

{ //

unpackedMatrix[i] = new int[size]; //

} //

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

{ //

for (int j = 0; j < size; ++j) //

{ //

unpackedMatrix[i][j] = 0; //

} //

} //

//

int i\_new = 0; //

int j\_new = 0; //

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

{ //

for (int j = 0; j < cols; ++j) //

{ //

if (packedMatrix[i][j] != 0) { //

i\_new = j; //

j\_new = j + index[i]; //

unpackedMatrix[i\_new][j\_new] = packedMatrix[i][j]; //

} //

} //

} //

//

return unpackedMatrix; //

}

int Summ\_elements(int\*\* Packed\_Matrix,int Count\_Digs, int cols, int summ) {

for (int i = 0; i < Count\_Digs; i++) {

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

if (((i + j) & 2) == 0) {

summ += Packed\_Matrix[i][j];

}

}

}

return summ;

}

int\*\* input\_and\_packMatrix(int\* rows1, int\* cols1, int\* size\_time, int\* Count\_Digs, int\* &Index\_Final) {

std::cout << "Укажите количество строк и столбцов упакованной матрицы: ";

int rows = 0, cols = 0, option = -1;

std::cin >> rows >> cols;

\*rows1 = rows; \*cols1 = cols;

int\*\* Matrix = new int\* [rows];

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

Matrix[i] = new int[cols];

}

while (option == -1) { // Выбор опции ввода.

option = chooseOption(); //

} //

if (option == 1) {

readFromConsole(Matrix, rows, cols);

}

else if (option == 2) {

readFromFile(Matrix, rows, cols);

}

else {

std::cout << "Надо было сразу нормально выбирать" << std::endl;

readFromConsole(Matrix, rows, cols);

}

list Index;

list Current\_Diagonal;

list Diagonals;

int count\_low = 0;

int count\_hight = 0;

//Нижние диагонали

for (int k = cols; k > 0; --k) {

int i = k;

int j = 0;

bool flag = false;

while (i < cols) {

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

flag = true;

Index.push\_back(j - i);

}

Current\_Diagonal.push\_back(Matrix[i][j]);

i += 1; j += 1;

}

if (flag) {

count\_low += 1;

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

Diagonals.push\_back(0);

}

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

Diagonals.push\_back(\*(Current\_Diagonal[l]));

}

Diagonals.push\_back(-999);

}

Current\_Diagonal.clear();

}

// Главная диагональ

int i = 0;

int j = 0;

bool flag = false;

while (i < cols) {

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

flag = true;

Index.push\_back(j - i);

}

Current\_Diagonal.push\_back(Matrix[i][j]);

i += 1; j += 1;

}

if (flag) {

for (int l = 0; l < cols; ++l) {

Diagonals.push\_back(\*(Current\_Diagonal[l]));

}

Diagonals.push\_back(-999);

}

Current\_Diagonal.clear();

//Верхние диагонали

for (int k = 1; k < cols; ++k) {

int i = 0;

int j = k;

bool flag = false;

while (i < cols - k) {

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

flag = true;

Index.push\_back(j - i);

}

Current\_Diagonal.push\_back(Matrix[i][j]);

i += 1; j += 1;

}

if (flag) {

count\_hight += 1;

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

Diagonals.push\_back(\*(Current\_Diagonal[l]));

}

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

Diagonals.push\_back(0);

}

Diagonals.push\_back(-999);

}

Current\_Diagonal.clear();

}

Index.remove\_duplicates();

int size = Index.len();

\*size\_time = size;

Index\_Final = new int[size];

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

Index\_Final[i] = \*(Index[i]);

}

int Count\_Digs2 = count\_low + count\_hight + 1;

\*Count\_Digs = Count\_Digs2;

int\*\* Packed\_Matrix1 = new int\* [Count\_Digs2];

for (int i = 0; i < Count\_Digs2; ++i) {

Packed\_Matrix1[i] = new int[cols];

}

Diagonals.values\_to\_matrix(Packed\_Matrix1, Count\_Digs2, cols);

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

delete[] Matrix[i];

}

delete[] Matrix;

return Packed\_Matrix1;

}

int main() {

setlocale(LC\_ALL, "RUSSIAN");

using namespace std;

int rows = 0, cols = 0, summ = 0, size = 0, Count\_Digs = 0;

int\* Index\_Final = new int[size];

int\*\* Packed\_Matrix = input\_and\_packMatrix(&rows, &cols, &size, &Count\_Digs, Index\_Final);

cout << '\n';

cout << "Запакованная матрица:" << endl;

printMatrix(Packed\_Matrix, Count\_Digs, cols);

cout << '\n';

cout << "Массив Индексов:" << endl;

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

cout << Index\_Final[i] << ' ';

}

cout << '\n'; cout << '\n';

cout << "Сумма элементов запакованной матрицы с четной суммой индексов:" << endl;

cout << Summ\_elements(Packed\_Matrix, Count\_Digs, cols, summ) << endl;

cout << '\n';

cout << "Распакованная матрица:" << endl;

int\*\* Unpacked\_Matrix = unpackMatrix(Packed\_Matrix, Index\_Final, Count\_Digs, cols);

printMatrix(Unpacked\_Matrix, cols, cols);

for (int i = 0; i < Count\_Digs; ++i) { // Очищаем память //

delete[] Packed\_Matrix[i]; //

}

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

delete[] Unpacked\_Matrix[i]; //

} //

delete[] Packed\_Matrix;

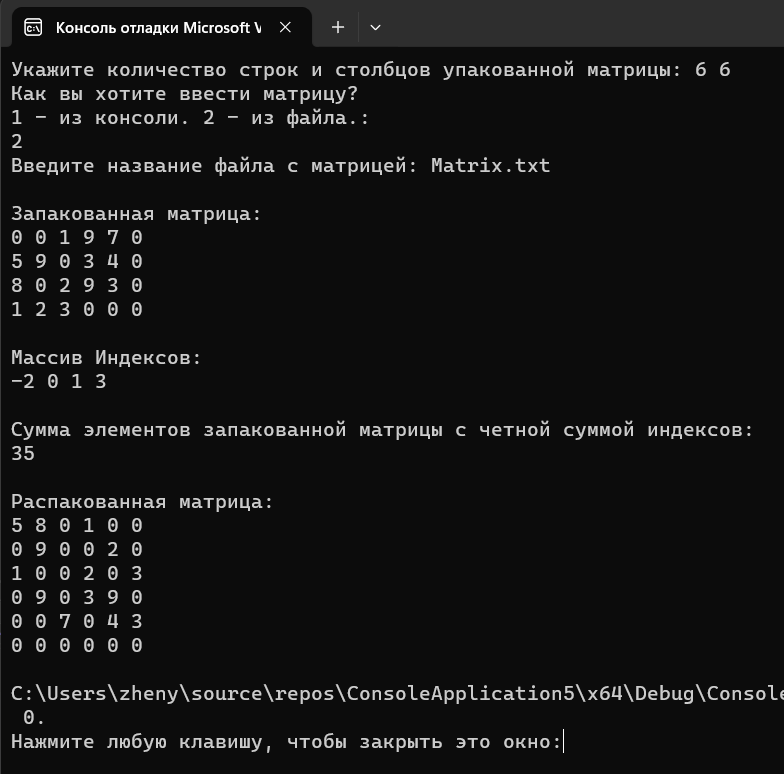
delete[] Index\_Final;

delete[] Unpacked\_Matrix; //

return 0;

}

Результат:



Вывод: Познакомились с разреженными матрицами и форматами их хранения.