Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ***ИУК «Информатика и управление»***

КАФЕДРА ***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

ЛАБОРАТОРНАЯ РАБОТА №4

«Обработка разреженных матриц»

ДИСЦИПЛИНА: «Типы и структуры данных»

Выполнил: студент гр. ИУК4-32Б _____ (_____Зудин Д.В._____)
(Подпись) (Ф.И.О.)

Проверил: _____ (___Пчелинцева Н.И.___)
(Подпись) (Ф.И.О.)

Дата сдачи (защиты): 29.12.22

Результаты сдачи (защиты):

- Балльная оценка: 4
- Оценка: хорошо

Калуга, 2022 г.

Цель: формирование практических навыков реализации хранения и обработки разреженных матриц.

Задачи:

1. Познакомиться с понятием разреженная/плотная матрица;
2. Изучить алгоритмы представления матриц в компактной форме;
3. Научиться реализовывать изученные алгоритмы средствами ООП-технологии;
4. Реализовать операцию над разреженными матрицами согласно варианту.

Вариант №3

Формулировка задания

Разреженные матрицы *A(n x m)* и *B(n x m)* хранятся в профильной схеме. Смоделировать операцию сложения двух матриц с получением результата в том же формате.

Листинг файла ProfileMatrix.h

```
#ifndef PMSS
#define PMSS
#include <vector>
#include <stdexcept>
#include <algorithm>

class ProfileMatrixStorageScheme
{
public:
    using Matrix = std::vector<std::vector<int>>;

    ProfileMatrixStorageScheme(const Matrix& matrix);
    ProfileMatrixStorageScheme(const std::vector<int>& AN, const
std::vector<int>& IA);
    Matrix getMatrix() const;
    std::vector<int> getAN() const;
    std::vector<int> getIA() const;
    friend ProfileMatrixStorageScheme operator+(const
ProfileMatrixStorageScheme& pmss1, const ProfileMatrixStorageScheme& pmss);

private:
    Matrix matrix;
    std::vector<int> AN;
    std::vector<int> IA;
};
#endif
```

Листинг файла ProfileMatrix.cpp

```
#include "ProfileMatrix.h"

ProfileMatrixStorageScheme::ProfileMatrixStorageScheme(const Matrix& _matrix)
{
    // проверка на симметричность
```

```
    for (auto i : _matrix)
    {
        if (i.size() != _matrix.size())
        {
            throw std::invalid_argument("Matrix must be symmetrical");
        }
    }

    const int N = _matrix.size(); // порядок матрицы

    for (size_t i = 0; i < N; i++)
    {
        for (size_t j = 0; j < N; j++)
        {
            if (_matrix[i][j] != _matrix[j][i])
            {
                throw std::invalid_argument("Matrix must be symmetrical");
            }
        }
    }

    matrix = _matrix;

    int cnt = 0;
    for (int i = 0; i < N; i++)
    {
        bool begin = false;
        for (int j = 0; j <= i; j++)
        {
            if (matrix[i][j] != 0)
            {
                begin = true;
            }
            if (begin)
            {
                cnt++;
                AN.push_back(matrix[i][j]);
            }
        }
        IA.push_back(cnt);
    }
}

ProfileMatrixStorageScheme::ProfileMatrixStorageScheme(const
std::vector<int>& _AN, const std::vector<int>& _IA)
{
    AN = _AN;
    IA = _IA;
    // конструируем матрицу matrix из AN и IA
    const int N = IA.size(); // порядок матрицы
    matrix = std::vector<std::vector<int>>(N, std::vector<int>(N));
    matrix[0][0] = AN[0];
    // заполним половину матрицы
    for (int i = 1; i < N; i++)
    {
        for (int j = IA[i - 1]; j < IA[i]; j++)
        {
            int m = IA[i] - IA[i - 1];    // количество элементов в i-ой
строке, начиная с первого ненулевого до диагонального включительно
            int jmini = i - m + 1;        // стобцовый индекс первого
ненулевого элемента в i-ой строке
            matrix[i][jmini + j - IA[i - 1]] = AN[j];
        }
```

```
    }
    // сделаем матрицу симметричной, заполнив вторую половину матрицы
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            if (i > j)
            {
                matrix[j][i] = matrix[i][j];
            }
        }
    }
}

ProfileMatrixStorageScheme::Matrix ProfileMatrixStorageScheme::getMatrix()
const
{
    return matrix;
}

std::vector<int> ProfileMatrixStorageScheme::getAN() const
{
    return AN;
}

std::vector<int> ProfileMatrixStorageScheme::getIA() const
{
    return IA;
}

ProfileMatrixStorageScheme operator+(const ProfileMatrixStorageScheme& pmssA,
const ProfileMatrixStorageScheme& pmssB)
{
    auto A = pmssA.getMatrix();
    auto B = pmssB.getMatrix();

    auto ANA = pmssA.getAN();
    auto IAA = pmssA.getIA();

    auto ANB = pmssB.getAN();
    auto IAB = pmssB.getIA();

    if (A.size() != B.size())
    {
        throw std::invalid_argument("Matrices must be of the same order");
    }

    const int N = A.size();
    std::vector<int> ANC;
    std::vector<int> IAC;
    ANC.push_back(ANA[0] + ANB[0]);
    for (int i = 1; i < N; i++)
    {
        std::vector<int> striA; // строка i матрицы A
        std::vector<int> striB; // строка i матрицы B

        for (int j = IAA[i - 1]; j < IAA[i]; j++)
        {
            striA.push_back(ANA[j]);
        }

        for (int j = IAB[i - 1]; j < IAB[i]; j++)
        {
```

```
            striB.push_back(ANB[j]);
        }

        std::vector<int> mnstri = (striA.size() > striB.size() ? striB :
striA); // вектор с минимальной длиной из striA и striB
        std::vector<int> mxstri = (striA.size() > striB.size() ? striA :
striB); // вектор с максимальной длиной из striA и striB

        std::reverse(mnstri.begin(), mnstri.end());
        std::reverse(mxstri.begin(), mxstri.end());

        for (int j = 0; j < mnstri.size(); j++)
        {
            mxstri[j] += mnstri[j];
        }
        std::reverse(mxstri.begin(), mxstri.end());
        for (auto j : mxstri)
        {
            ANC.push_back(j);
        }
        IAC.push_back(ANC.size());
    }
    IAC.insert(IAC.begin(), 1);
    ProfileMatrixStorageScheme pmssC(ANC, IAC);
    return pmssC;
}
```

Листинг файла main.cpp

```
#include <iostream>
#include <ctime>
#include <random>
#include "ProfileMatrix.h"
#include <sstream>
#include <iomanip>
#include <algorithm>

template <typename T>
std::ostream& operator<<(std::ostream& out, const std::vector<T>& v)
{
    out << "[";
    for (size_t i = 0; i < v.size(); i++)
    {
        out << v[i] << (i == v.size() - 1 ? "" : ", ");
    }
    out << "]";
    return out;
}

template <typename T>
std::ostream& operator<<(std::ostream& out, const
std::vector<std::vector<T>>& vv)
{
    int max_length = 0;
    for (auto v : vv)
    {
        max_length = std::to_string(*std::max_element(v.begin(),
v.end())).length();
    }
    max_length += 2;
    for (auto v : vv)
    {
```

```
        for (auto i : v)
        {
            std::cout << std::setw(max_length) << i;
        }
        std::cout << "\n";
    }
    return out;
}

inline int getRandom(const int _min, const int _max)
{
    return rand() % (_max - _min + 1) + _min;
}

std::vector<std::vector<int>> getBandMatrix(const int N)
{
    std::vector<std::vector<int>> matrix(N, std::vector<int>(N));
    for (int i = 0; i < N; i++)
    {
        for (int j = 0; j < N; j++)
        {
            int bandwidth = getRandom(1, N / 2);
            if (i >= j)
            {
                matrix[i][j] = matrix[j][i] = (j <= i - bandwidth ? 0 :
rand() % 100);
                if (rand() % 5 == 0 && i != j)
                {
                    matrix[i][j] = matrix[j][i] = 0;
                }
            }
        }
    }
    return matrix;
}

int main()
{
    srand(time(nullptr));
    ProfileMatrixStorageScheme::Matrix matrixA = getBandMatrix(10);
    ProfileMatrixStorageScheme::Matrix matrixB = getBandMatrix(10);
    ProfileMatrixStorageScheme pmssA(matrixA);
    ProfileMatrixStorageScheme pmssB(matrixB);

    std::cout << "matrix A = \n" << pmssA.getMatrix();
    std::cout << "ANA = " << pmssA.getAN() << "\n";
    std::cout << "IAA = " << pmssA.getIA() << "\n\n";

    std::cout << "matrix B = \n" << pmssB.getMatrix();
    std::cout << "ANB = " << pmssB.getAN() << "\n";
    std::cout << "IAB = " << pmssB.getIA() << "\n\n";

    ProfileMatrixStorageScheme pmssC = pmssA + pmssB;
    std::cout << "matrix C = A + B =\n" << pmssC.getMatrix();
    std::cout << "ANC = " << pmssC.getAN() << "\n";
    std::cout << "IAC = " << pmssC.getIA() << "\n";

    return 0;
}
```

Результат выполнения программы

```
matrix A =
  51  89   0  18   0   0   0   0   0   0
  89  75  79   0   0   0   0   0   0   0
   0  79  20  68   0   4   0   0   0   0
  18   0  68  81   0   0   0   0   0   0
   0   0   0   0  36  23  76   0   0   0
   0   0   4   0  23  81  80   0   0   0
   0   0   0   0  76  80  23   5  91   0
   0   0   0   0   0   0   5   6  42  25
   0   0   0   0   0   0  91  42  87   0
   0   0   0   0   0   0   0  25   0  50
ANA = [51, 89, 75, 79, 20, 18, 0, 68, 81, 36, 4, 0, 23, 81, 76, 80, 23, 5, 6,
91, 42, 87, 25, 0, 50]
IAA = [1, 3, 5, 9, 10, 14, 17, 19, 22, 25]

matrix B =
   8  35  21  73   0   0   0   0   0   0
  35   8  32  66  31   0   0   0   0   0
  21  32  75   0  54   0   0   0   0   0
  73  66   0  29   0  41   0   0   0   0
   0  31  54   0  16  71  13   1   0   0
   0   0   0  41  71  75   0  65  76   0
   0   0   0   0  13   0  57  40   0   0
   0   0   0   0   1  65  40  70  69  22
   0   0   0   0   0  76   0  69  72   0
   0   0   0   0   0   0   0  22   0  38
ANB = [8, 35, 8, 21, 32, 75, 73, 66, 0, 29, 31, 54, 0, 16, 41, 71, 75, 13, 0,
57, 1, 65, 40, 70, 76, 0, 69, 72, 22, 0, 38]
IAB = [1, 3, 6, 10, 14, 17, 20, 24, 28, 31]

matrix C = A + B =
  59 124  21  91   0   0   0   0   0   0
 124  83 111  66  31   0   0   0   0   0
  21 111  95  68  54   4   0   0   0   0
  91  66  68 110   0  41   0   0   0   0
   0  31  54   0  52  94  89   1   0   0
   0   0   4  41  94 156  80  65  76   0
   0   0   0   0  89  80  80  45  91   0
   0   0   0   0   1  65  45  76 111  47
   0   0   0   0   0  76  91 111 159   0
   0   0   0   0   0   0   0  47   0  88
```

```
ANC = [59, 124, 83, 21, 111, 95, 91, 66, 68, 110, 31, 54, 0, 52, 4, 41, 94,
156, 89, 80, 80, 1, 65, 45, 76, 76, 91, 111, 159, 47, 0, 88]
IAC = [1, 3, 6, 10, 14, 18, 21, 25, 29, 32]
```

Выводы:

В ходе работы были сформированы практические навыки реализации хранения и обработки разреженных матриц.