**Московский Авиационный институт**

**(Национальный исследовательский университет)**

**Факультет №8**

**«Компьютерные науки и прикладная математика»**

**Кафедра 806**

**«Вычислительная математика и программирование»**

**Курсовая работа**

по теме

«Разреженные матрицы»

|  |  |
| --- | --- |
| Студент: | Концебалов О.С. |
| Группа: | М8О-109Б-22 |
| Преподаватель: | Сысоев М. А. |
| Подпись: |  |
| Оценка: |  |

Москва, 2023

**Задание**

Составить программу на языке Си с функциями для обработки прямоугольных разреженных матриц с элементами целого типа, которая:

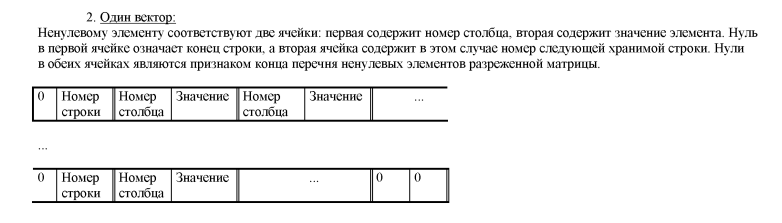
Вводит матрицы различного размера с одновременным размещением ненулевых элементов в разреженной матрице в соответствии с заданной схемой;

Печатает введенные матрицы во внутреннем представлении и в обычном виде; выполняет необходимые преобразования разреженных матриц (или вычисления над ними) путем обращения к соответствующим функциям;

Печатает результат преобразования во внутреннем представлении и в обычном виде.

В процедурах и функциях предусмотреть проверки и печать сообщений в случаях ошибок в задании параметров. Для отладки использовать матрицы, содержащие 5- 10% ненулевых элементов, с максимальным числом элементов 100.

**Вариант схемы размещения матрицы:**



**Вариант преобразования:**

5. Умножить вектор-строку на разреженную матрицу и вычислить количество ненулевых элементов результата

**Общий метод решения**

Разреженная матрица — это матрица с преимущественно нулевыми элементами. В противном случае, если большая часть элементов матрицы ненулевые, матрица считается плотной.

Хранение разреженной матрицы в памяти должно обеспечивать:

1. экономию памяти
2. быстрый доступ к нулевым и ненулевым элементам по их индексу.

Поэтому, хранение разреженной матрицы с помощью одного вектора крайне удобно. Мы представляем исходную матрицу M размеров m x n, содержащую некоторое число ненулевых значений в виде одного вектора. В нем каждому элементу соответствуют две ячейки – первая это номер столбца, а вторая значение элемента. Если в первой ячейке ноль, то строка закончилась, а во второй будет номер следующей строки. Нули в обеих ячейках означают, что ненулевые элементы матрицы кончились.

**Структура:**

1. Класс Vector, похожий на std::vector из стандартной библиотеки, хранящий шаблонный тип.
2. Класс Matrix, предназначенный для работы с разреженными матрицами.
3. main - программа для интерактивной работы.
4. benchmark - тест-сравнение времени работы стандартного и пользовательского вектора.

**Алгоритм решения поставленной задачи**

Для обработки разреженных матриц опишем класс вектора с его множеством операций и реализуем вектор на С++. Отдельно опишем класс для обработки разреженных матриц:

1. Считывание матриц в обычном виде из файла с преобразованием в векторы согласно заданной схеме размещения.
2. Печать матрицы в естественном виде.
3. Печать вектора (схема размещения ненулевых элементов разреженной матрицы).
4. Выполнение заданного преобразования

**Код программы**

**myVector.hpp**

#ifndef MYVECTORHPP

#define MYVECTORHPP

#include <iostream>

template <typename T>

class myVector{

private:

    T\* data;

    size\_t vec\_size;

    size\_t vec\_capacity;

public:

    myVector();

    myVector(const myVector<T>& another\_vector);

    myVector(const std::initializer\_list<T>& list);

    ~myVector();

    void reserve(size\_t n);

    void resize(size\_t new\_size, const T& value = T());

    void shrink\_to\_fit();

    void push\_back(const T& value);

    void pop\_back();

    T& at(size\_t index);

    const T& front() const;

    const T& back() const;

    bool empty() const;

    size\_t size() const;

    size\_t capacity() const;

    void clear();

    template <typename... Args>

    void emplace\_back(const Args& ...args);

    T& operator[](size\_t i) const;

    bool operator==(const myVector<T>& another\_vector);

    bool operator!=(const myVector<T>& another\_vector);

    void erase(size\_t element\_index);

    void erase(size\_t start\_index, size\_t end\_index);

    void swap(myVector<T>& another\_vector);

};

#endif

**myVector.cpp**

#include "MyVector.hpp"

#include <climits>

#include <memory>

template <typename T>

myVector<T>::myVector(){

    data = reinterpret\_cast<T\*>(new int8\_t[sizeof(T)]);

    vec\_size = 0;

    vec\_capacity = 1;

}

template <typename T>

myVector<T>::myVector(const myVector<T>& another\_vector){

    resize(another\_vector.size());

    for (size\_t i = 0; i != vec\_size; ++i){

        data[i].~T();

        new (data + i) T(another\_vector[i]);

    }

}

template <typename T>

myVector<T>::myVector(const std::initializer\_list<T>& list){

    T\* new\_data = reinterpret\_cast<T\*>(new int8\_t[list.size() \* sizeof(T)]);

    try{

        std::uninitialized\_copy(list.begin(), list.end(), new\_data);

    } catch (...){

        delete[] reinterpret\_cast<int8\_t\*>(new\_data);

        throw;

    }

    delete[] reinterpret\_cast<int8\_t\*>(data);

    data = new\_data;

    vec\_size = list.size();

    vec\_capacity = list.size();

}

template <typename T>

myVector<T>::~myVector(){

    for (size\_t i = 0; i != vec\_size; ++i){

        data[i].~T();

    }

    delete[] reinterpret\_cast<int8\_t\*>(data);

}

template <typename T>

void myVector<T>::reserve(size\_t reserved\_size){

    if (reserved\_size > ULLONG\_MAX){

        throw std::range\_error("Capacity overflow");

    }

    if (reserved\_size <= vec\_capacity) return;

    T\* new\_data = reinterpret\_cast<T\*>(new int8\_t[reserved\_size \* sizeof(T)]);

    try{

        std::uninitialized\_copy(data, data + vec\_size, new\_data);

    } catch(...){

        delete[] reinterpret\_cast<int8\_t\*>(new\_data);

        throw;

    }

    for (size\_t i = 0; i != vec\_size; ++i){

        data[i].~T();

    }

    delete[] reinterpret\_cast<int8\_t\*>(data);

    data = new\_data;

    vec\_capacity = reserved\_size;

}

template <typename T>

void myVector<T>::resize(size\_t new\_size, const T& value){

    if (new\_size > ULLONG\_MAX){

        throw std::range\_error("Capacity overflow");

    }

    if (new\_size > vec\_capacity) reserve(new\_size);

    if (new\_size > vec\_size){

        for (size\_t i = vec\_size; i != new\_size; ++i){

            new (data + i) T(value);

        }

    }

    if (new\_size < vec\_size){

        for (size\_t i = new\_size; i != vec\_size; ++i){

            data[i].~T();

        }

    }

    vec\_size = new\_size;

}

template <typename T>

void myVector<T>::shrink\_to\_fit(){

    if (vec\_size < vec\_capacity){

        T\* new\_data = reinterpret\_cast<T\*>(new int8\_t[vec\_size \* sizeof(T)]);

        for (size\_t i = 0; i != vec\_size; ++i){

            new (new\_data + i) T(data[i]);

        }

        delete[] reinterpret\_cast<int8\_t\*>(data);

        data = new\_data;

        vec\_capacity = vec\_size;

    }

}

template <typename T>

void myVector<T>::push\_back(const T& value){

    if (vec\_capacity == vec\_size){

        if (vec\_capacity \* 2 < vec\_capacity){

            throw std::range\_error("Capacity overflow");

        }

        reserve(2 \* vec\_capacity);

    }

    new (data + vec\_size) T(value);

    ++vec\_size;

}

template <typename T>

void myVector<T>::pop\_back(){

    if (vec\_size == 0) return;

    --vec\_size;

    data[vec\_size].~T();

}

template <typename T>

T& myVector<T>::at(size\_t index){

    if (index >= vec\_size){

        throw std::out\_of\_range("Index out of range");

    }

    return data[index];

}

template <typename T>

const T& myVector<T>::front() const{

    if (vec\_size == 0){

        throw std::range\_error("Vector is empty");

    }

    return data[0];

}

template <typename T>

const T& myVector<T>::back() const{

    if (vec\_size == 0){

        throw std::range\_error("Vector is empty");

    }

    return data[vec\_size - 1];

}

template <typename T>

bool myVector<T>::empty() const{

    return vec\_size == 0;

}

template <typename T>

size\_t myVector<T>::size() const{

    return vec\_size;

}

template <typename T>

size\_t myVector<T>::capacity() const{

    return vec\_capacity;

}

template <typename T>

void myVector<T>::clear(){

    for (size\_t i = 0; i != vec\_size; ++i){

        data[i].~T();

    }

    vec\_size = 0;

}

template <typename T>

template <typename... Args>

void myVector<T>::emplace\_back(const Args& ...args){

    if (vec\_capacity == vec\_size) reserve(2 \* vec\_capacity);

    new (data + vec\_size) T(args...);

    ++vec\_size;

}

template <typename T>

T& myVector<T>::operator[](size\_t i) const{

    return data[i];

}

template <typename T>

bool myVector<T>::operator==(const myVector<T>& another\_vector){

    if (vec\_size != another\_vector.vec\_size) return false;

    for (size\_t i = 0; i != vec\_size; ++i){

        if (this->data[i] != another\_vector.data[i]) return false;

    }

    return true;

}

template <typename T>

bool myVector<T>::operator!=(const myVector<T>& another\_vector){

    if (vec\_size != another\_vector.vec\_size) return true;

    for (size\_t i = 0; i != vec\_size; ++i){

        if (this->data[i] != another\_vector.data[i]) return true;

    }

    return false;

}

template <typename T>

void myVector<T>::erase(size\_t element\_index){

    if (element\_index >= vec\_size){

        throw std::out\_of\_range("Index out of range");

    }

    for (size\_t i = element\_index; i != vec\_size - 1; ++i){

        data[i].~T();

        data[i] = data[i + 1];

    }

    --vec\_size;

    resize(vec\_size);

}

template <typename T>

void myVector<T>::erase(size\_t start, size\_t end){

    if (start > end){

        throw std::invalid\_argument("Start index must be less than end index");

    }

    if (start > vec\_size || end > vec\_size){

        throw std::out\_of\_range("Index out of range");

    }

    size\_t to\_delet = end - start + 1;

    for (size\_t i = start; i != vec\_size - to\_delet; ++i){

        data[i].~T();

        data[i] = data[i + to\_delet];

    }

    vec\_size -= to\_delet;

    resize(vec\_size);

}

template <typename T>

void myVector<T>::swap(myVector<T>& another\_vector){

    T\* temp\_data = data;

    data = another\_vector.data;

    another\_vector.data = temp\_data;

    size\_t temp\_size = vec\_size;

    vec\_size = another\_vector.vec\_size;

    another\_vector.vec\_size = temp\_size;

    size\_t temp\_capacity = vec\_capacity;

    vec\_capacity = another\_vector.vec\_capacity;

    another\_vector.vec\_capacity = temp\_capacity;

}

**Matrix.hpp**

#ifndef MATRIXHPP

#define MATRIXHPP

#include "MyVector.hpp"

#include <iostream>

class Matrix{

private:

    myVector<int> line\_matrix;

    size\_t matrix\_lines;

    size\_t matrix\_columns;

public:

    Matrix();

    Matrix(size\_t lines, size\_t columns);

    friend std::istream& operator>>(std::istream& is, Matrix& matrix);

    friend std::ostream& operator<<(std::ostream& os, Matrix& matrix);

    myVector<int> multiply(const myVector<int>& vector\_string);

    void print();

};

#endif

**Matrix.cpp**

#include "Matrix.hpp"

#include <iostream>

Matrix::Matrix(): matrix\_lines(0), matrix\_columns(0){

    line\_matrix.push\_back(0);

    line\_matrix.push\_back(0);

}

Matrix::Matrix(size\_t lines, size\_t columns): matrix\_lines(lines), matrix\_columns(columns){

    line\_matrix.push\_back(0);

    line\_matrix.push\_back(0);

}

std::istream& operator>>(std::istream& is, Matrix& matrix){

    matrix.line\_matrix.pop\_back();

    size\_t line, columns;

    int input\_value;

    is >> line >> columns;

    matrix.matrix\_lines = line;

    matrix.matrix\_columns = columns;

    for (size\_t i = 0; i != line; ++i){

        for (size\_t j = 0; j != columns; ++j){

            is >> input\_value;

            if (input\_value != 0){

                if (matrix.line\_matrix.back() == 0){

                    matrix.line\_matrix.push\_back(i + 1);

                }

                matrix.line\_matrix.push\_back(j + 1);

                matrix.line\_matrix.push\_back(input\_value);

            }

        }

        if (matrix.line\_matrix.back() != 0){

            matrix.line\_matrix.push\_back(0);

        }

    }

    matrix.line\_matrix.push\_back(0);

    return is;

}

std::ostream& operator<<(std::ostream& os, Matrix& matrix){

    for (size\_t i = 0; i != matrix.line\_matrix.size(); ++i){

        os << matrix.line\_matrix[i] << " ";

    }

    return os;

}

myVector<int> Matrix::multiply(const myVector<int>& vector\_string){

    if (vector\_string.size() != matrix\_lines){

        throw std::range\_error("Vector columns not equal matrix lines");

    }

    myVector<int> result;

    for (size\_t i = 0; i != matrix\_columns; ++i){

        result.push\_back(0);

    }

    size\_t cur\_line = 0;

    while (cur\_line < line\_matrix.size()){

        int j = line\_matrix[cur\_line];

        if (j == 0){

            ++cur\_line;

            if (cur\_line >= line\_matrix.size() || line\_matrix[cur\_line] == 0) break;

            j = line\_matrix[cur\_line];

        }

        int cur\_column = cur\_line + 1;

        int column\_of\_result\_vector = line\_matrix[cur\_line] - 1;

        for (size\_t column = cur\_column; line\_matrix[column] != 0; column += 2){

            int value = line\_matrix[column + 1];

            result[line\_matrix[column] - 1] += (vector\_string[column\_of\_result\_vector] \* value);

            cur\_line = column;

        }

        ++cur\_line;

    }

    uint64\_t number\_of\_non\_zero\_elements = 0;

    std::cout << "Result: ( ";

    for (size\_t i = 0; i != result.size(); ++i){

        if (result[i] != 0) ++number\_of\_non\_zero\_elements;

        std::cout << result[i] << " ";

    }

    std::cout << ")\nNumber of non-zeros elements of the result: " << number\_of\_non\_zero\_elements << "\n";

    return result;

}

void Matrix::print(){

    size\_t line = 1, column = 1;

    for (size\_t i = 0; i < line\_matrix.size(); ++i) {

        if (!line\_matrix[i]) {

            ++i;

            if (!line\_matrix[i]) {

                while (line <= matrix\_lines) {

                    while (column <= matrix\_columns) {

                        column++;

                        std::cout << 0 << " ";

                    }

                    std::cout << '\n';

                    line++;

                    column = 1;

                }

                return;

            }

            while (line < line\_matrix[i]) {

                while (column <= matrix\_columns) {

                    column++;

                    std::cout << 0 << " ";

                }

                std::cout << '\n';

                line++;

                column = 1;

            }

        }

        else {

            while (column != line\_matrix[i]) {

                column++;

                std::cout << 0 << " ";

            }

            ++i;

            column++;

            std::cout << line\_matrix[i] << " ";

        }

    }

}

**Benchmark.cpp**

#include "MyVector.hpp"

#include <algorithm>

#include <chrono>

#include <fstream>

#include <iostream>

#include <vector>

void benchmark(){

    std::ios\_base::sync\_with\_stdio(false);

    std::cin.tie(nullptr);

    std::cout.tie(nullptr);

    std::ifstream tests\_file("..\\test.txt", std::ios::in);

    if (!tests\_file.is\_open()){

        std::cerr << "ERROR OPENING FILE\n";

        return;

    }

    myVector<int> my\_vector;

    std::vector<int> original\_vector;

    int amount\_of\_numbers, number;

    std::cout << "\nComparing \"myVector\" and the \"vector\" from STL\n\n";

    tests\_file >> amount\_of\_numbers;

    std::cout << "Amount of numbers: " << amount\_of\_numbers << "\n\n";

    std::cout << "push\_back, complexity O(1)\n";

    std::chrono::steady\_clock::time\_point start\_time = std::chrono::steady\_clock::now();

    for (int i = 0; i != amount\_of\_numbers; ++i){

        tests\_file >> number;

        my\_vector.push\_back(number);

    }

    std::chrono::steady\_clock::time\_point end\_time = std::chrono::steady\_clock::now();

    std::cout << "Time of work my\_vector: " <<

        std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time).count() << " milliseconds\n";

    tests\_file.clear();

    tests\_file.seekg(0);

    tests\_file >> number;

    start\_time = std::chrono::steady\_clock::now();

    for (int i = 0; i != amount\_of\_numbers; ++i){

        tests\_file >> number;

        original\_vector.push\_back(number);

    }

    end\_time = std::chrono::steady\_clock::now();

    std::cout << "Time of work original\_vector: " <<

        std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time).count() << " milliseconds\n";

    if (my\_vector.size() != original\_vector.size()){

        throw std::exception();

    }

    for (size\_t i = 0; i != original\_vector.size(); ++i){

        if (my\_vector[i] != original\_vector[i]){

            std::cout << "ERROR: index " << i << " my\_vector[i] = " << my\_vector[i] <<

                    ", original\_vector[i] = " << original\_vector[i] << "\n";

            throw std::exception();

        }

    }

    std::cout << "pop\_back, complexity O(1)\n";

    start\_time = std::chrono::steady\_clock::now();

    for (int i = 0; i != amount\_of\_numbers; ++i){

        my\_vector.pop\_back();

    }

    end\_time = std::chrono::steady\_clock::now();

    std::cout << "Time of work my\_vector: " <<

        std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time).count() << " milliseconds\n";

    start\_time = std::chrono::steady\_clock::now();

    for (int i = 0; i != amount\_of\_numbers; ++i){

        original\_vector.pop\_back();

    }

    end\_time = std::chrono::steady\_clock::now();

    std::cout << "Time of work original\_vector: " <<

        std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time).count() << " milliseconds\n";

    if (my\_vector.size() != original\_vector.size()){

        throw std::exception();

    }

    for (size\_t i = 0; i != original\_vector.size(); ++i){

        if (my\_vector[i] != original\_vector[i]){

            std::cout << "ERROR: index " << i << " my\_vector[i] = " << my\_vector[i] <<

                    ", original\_vector[i] = " << original\_vector[i] << "\n";

            throw std::exception();

        }

    }

    std::cout << "shrink\_to\_fit\n";

    start\_time = std::chrono::steady\_clock::now();

    for (int i = 0; i != amount\_of\_numbers; ++i){

        my\_vector.shrink\_to\_fit();

    }

    end\_time = std::chrono::steady\_clock::now();

    std::cout << "Time of work my\_vector: " <<

        std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time).count() << " milliseconds\n";

    start\_time = std::chrono::steady\_clock::now();

    for (int i = 0; i != amount\_of\_numbers; ++i){

        original\_vector.shrink\_to\_fit();

    }

    end\_time = std::chrono::steady\_clock::now();

    std::cout << "Time of work original\_vector: " <<

        std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time).count() << " milliseconds\n";

    if (my\_vector.capacity() != original\_vector.capacity()){

        std::cout << "ERROR: my\_vector.capacity() = " << my\_vector.capacity() <<

                ", original\_vector.capacity() = " << original\_vector.capacity() << "\n";

    }

    std::cout << "reserve\n";

    start\_time = std::chrono::steady\_clock::now();

    my\_vector.reserve(2 \* amount\_of\_numbers);

    end\_time = std::chrono::steady\_clock::now();

    std::cout << "Time of work my\_vector: " <<

        std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time).count() << " milliseconds\n";

    start\_time = std::chrono::steady\_clock::now();

    original\_vector.reserve(2 \* amount\_of\_numbers);

    end\_time = std::chrono::steady\_clock::now();

    std::cout << "Time of work original\_vector: " <<

        std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time).count() << " milliseconds\n";

    std::cout << "my\_vector.capacity() = " << my\_vector.capacity() <<

        ", original\_vector.capacity() = " << original\_vector.capacity() << "\n";

    std::cout << "resize\n";

    start\_time = std::chrono::steady\_clock::now();

    my\_vector.resize(2 \* amount\_of\_numbers, number);

    end\_time = std::chrono::steady\_clock::now();

    std::cout << "Time of work my\_vector: " <<

        std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time).count() << " milliseconds\n";

    start\_time = std::chrono::steady\_clock::now();

    original\_vector.resize(2 \* amount\_of\_numbers, number);

    end\_time = std::chrono::steady\_clock::now();

    std::cout << "Time of work original\_vector: " <<

        std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time).count() << " milliseconds\n";

    if (my\_vector.size() != original\_vector.size()){

        throw std::exception();

    }

    for (size\_t i = 0; i != original\_vector.size(); ++i){

        if (my\_vector[i] != original\_vector[i]){

            std::cout << "ERROR: index " << i << " my\_vector[i] = " << my\_vector[i] <<

                ", original\_vector[i] = " << original\_vector[i] << "\n";

            throw std::exception();

        }

    }

    std::cout << "my\_vector.capacity() = " << my\_vector.capacity() <<

        ", original\_vector.capacity() = " << original\_vector.capacity() << "\n";

    std::cout << "my\_vector.size() = " << my\_vector.size() <<

        ", original\_vector.size() = " << original\_vector.size() << "\n";

    std::cout << "clear\n";

    start\_time = std::chrono::steady\_clock::now();

    my\_vector.clear();

    end\_time = std::chrono::steady\_clock::now();

    std::cout << "Time of work my\_vector: " <<

        std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time).count() << " milliseconds\n";

    start\_time = std::chrono::steady\_clock::now();

    original\_vector.clear();

    end\_time = std::chrono::steady\_clock::now();

    std::cout << "Time of work original\_vector: " <<

        std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time).count() << " milliseconds\n";

    if (my\_vector.size() != 0 && original\_vector.size() != 0){

        throw std::exception();

    }

    std::cout << "my\_vector.capacity() = " << my\_vector.capacity() <<

        ", original\_vector.capacity() = " << original\_vector.capacity() << "\n";

    std::cout << "my\_vector.size() = " << my\_vector.size() <<

        ", original\_vector.size() = " << original\_vector.size() << "\n";

    std::cout << "copy constructor\n";

    start\_time = std::chrono::steady\_clock::now();

    myVector<int> my\_vector1(my\_vector);

    end\_time = std::chrono::steady\_clock::now();

    std::cout << "Time of work my\_vector: " <<

        std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time).count() << " milliseconds\n";

    start\_time = std::chrono::steady\_clock::now();

    std::vector<int> original\_vector1(original\_vector);

    end\_time = std::chrono::steady\_clock::now();

    std::cout << "Time of work original\_vector: " <<

        std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time).count() << " milliseconds\n";

    std::cout << "equality\n";

    start\_time = std::chrono::steady\_clock::now();

    std::cout << (my\_vector == my\_vector1) << "\n";

    end\_time = std::chrono::steady\_clock::now();

    std::cout << "Time of work my\_vector: " <<

        std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time).count() << " milliseconds\n";

    start\_time = std::chrono::steady\_clock::now();

    std::cout << (original\_vector == original\_vector1) << "\n";

    end\_time = std::chrono::steady\_clock::now();

    std::cout << "Time of work original\_vector: " <<

        std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time).count() << " milliseconds\n";

    std::cout << "swap\n";

    start\_time = std::chrono::steady\_clock::now();

    my\_vector.swap(my\_vector1);

    end\_time = std::chrono::steady\_clock::now();

    std::cout << "Time of work my\_vector: " <<

        std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time).count() << " milliseconds\n";

    start\_time = std::chrono::steady\_clock::now();

    original\_vector.swap(original\_vector1);

    end\_time = std::chrono::steady\_clock::now();

    std::cout << "Time of work original\_vector: " <<

        std::chrono::duration\_cast<std::chrono::milliseconds>(end\_time - start\_time).count() << " milliseconds\n";

    return;

}

**Run.cpp**

#include "MyVector.hpp"

#include "MyVector.cpp"

#include "benchmark.cpp"

#include "Matrix.hpp"

#include "Matrix.cpp"

#include <iostream>

int main(){

    myVector<int> vector\_string = {4, 10, 52, 0, 34, 47, -4, 9};

    Matrix sparse\_matrix;

    std::cin >> sparse\_matrix;

    myVector<int> res = sparse\_matrix.multiply(vector\_string);

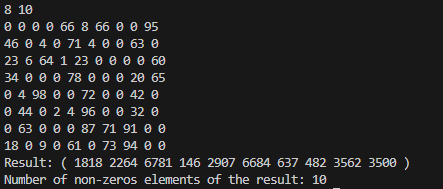
    benchmark();

    return 0;

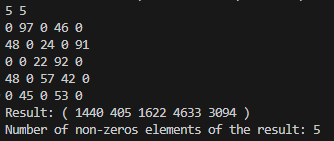
}

**Тестирование**

**1) vector\_string = {4, 10, 52, 0, 34, 47, -4, 9};**

****

**2) vector\_string = {0, 34, 47, -4, 9};**

****

**Тесты производительности**

Количество чисел для сравнения: 1751438

**push\_back, complexity O(1)**

Time of work my\_vector: 695 milliseconds

Time of work original\_vector: 751 milliseconds

**pop\_back, complexity O(1)**

Time of work my\_vector: 2 milliseconds

Time of work original\_vector: 6 milliseconds

**shrink\_to\_fit**

Time of work my\_vector: 4 milliseconds

Time of work original\_vector: 10 milliseconds

**reserve**

Time of work my\_vector: 0 milliseconds

Time of work original\_vector: 0 milliseconds

my\_vector.capacity() = 3502876

original\_vector.capacity() = 3502876

**resize**

Time of work my\_vector: 8 milliseconds

Time of work original\_vector: 3 milliseconds

my\_vector.capacity() = 3502876

original\_vector.capacity() = 3502876

my\_vector.size() = 3502876

original\_vector.size() = 3502876

**clear**

Time of work my\_vector: 2 milliseconds

Time of work original\_vector: 0 milliseconds

my\_vector.capacity() = 3502876

original\_vector.capacity() = 3502876

my\_vector.size() = 0

original\_vector.size() = 0

**copy constructor**

Time of work my\_vector: 0 milliseconds

Time of work original\_vector: 0 milliseconds

**equality**

Time of work my\_vector: 0 milliseconds

Time of work original\_vector: 0 milliseconds

**swap**

Time of work my\_vector: 0 milliseconds

Time of work original\_vector: 0 milliseconds

Как мы видим из результатов тестирования, мой кастомный вектор работает быстрее вектора из STL примерно в 1,5-2 раза (удивился). Думаю это связано с тем, что наши реализации очень похоже друг на друга, но вектор из STL будет использовать более безопасные и оптимальные функции и методы, чем мой. Также вектор из STL написан полностью на аллокаторах и использует мув семантику, а я их в своей реализации не использовал, может быть это тоже как-то влияет.

Значительные отличия по скорости отличаются методы push\_back и pop\_back, а также shrink\_to\_fit и clear. Shrink\_to\_fit осуществляет вызов деструктора объектов в цикле, что может занимать значительное время при большом количестве элементов, в стандартном векторе реализация метода скорее всего неким образом более оптимизирована. В pop\_back я просто декрементирую size и вызываю деструктор элемента по индексу size, видимо в STL векторе это реализовано как-то иначе. В push\_back я делаю placement new нового элемента, предварительно проверив capacity и size. Предположительно мой работает быстрее, потому что различается время работы reserve стандартного вектора и моего.

Еще стоит отметить различия во времени работы resize. Мой работает медленнее возможно потому что я вызываю деструкторы от объектов в цикле, что может занимать довольно много времени.

**Заключение**

В результате получилось выполнить поставленное задание — реализовал оптимальное хранение разреженной матрицы и написал функцию для выполнения умножения вектора-строки на матрицу.

В ходе работы написал свой вектор, который оказался даже побыстрее чем вектор из STL (шок), что было очень интересно. Благодаря этому я получил хорошие навыки работы с динамическими структурами и опыт работы с памятью, которые очень пригодятся мне в дальнейшем.