

**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:	Ефименко К. И.
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:

#### 2. Один вектор:

Ненулевому элементу соответствуют две ячейки: первая содержит номер столбца, вторая содержит значение элемента. Ноль в первой ячейке означает конец строки, а вторая ячейка содержит в этом случае номер следующей хранимой строки. Нули в обеих ячейках являются признаком конца перечня ненулевых элементов разреженной матрицы.

0	Номер строки	Номер столбца	Значение	Номер столбца	Значение	...
---	--------------	---------------	----------	---------------	----------	-----

...

0	Номер строки	Номер столбца	Значение	...	0	0
---	--------------	---------------	----------	-----	---	---

### 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  $m \times n$ , 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 C++. 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

#### Vector.hpp

```
#ifndef VECTOR_HPP_INCLUDED
#define VECTOR_HPP_INCLUDED

#include <iostream>

template<typename T>
class Vector {
private:
    size_t capacity = 0;
    size_t size = 0;
    T* array = nullptr;

public:
    Vector();
    Vector(size_t);
    Vector(const std::initializer_list<T>&);
    Vector(const Vector&);
    Vector& operator=(const Vector<T>&);
    ~Vector();

    size_t get_size() const;
    size_t get_capacity() const;
};
```

```

bool empty() const;
bool operator==(const Vector<T>&) const;
bool operator!=(const Vector<T>&) const;

void resize(size_t, const T& = T());
void push_back(const T&);
void reserve(size_t n);
void pop_back();
void clear();
void shrink_to_fit();

template <typename... Args>
void emplace_back(const Args& ...args);

T& operator[](size_t);
T& at(size_t);
T& front();
T& back();
const T& operator[](size_t) const;
const T& at(size_t) const;
const T& back() const;
const T& front() const;
};

#include "../src/Vector.cpp"

#endif

```

## Vector.cpp

```

#include "../include/Vector.hpp"
#include <memory>
#include <climits>

template <typename T>
Vector<T>::Vector() {
    size = 0;
    capacity = 1;
    array = new T[1];
}

template <typename T>
Vector<T>::Vector(size_t capacity) {
    size = 0;
    this->capacity = capacity;
    array = reinterpret_cast<T*>(new int8_t[capacity * sizeof(T)]);
}

template <typename T>
Vector<T>::Vector(const Vector& vec): Vector(vec.capacity) {
    memcpy(array, vec.array, capacity * sizeof(T));
}

template <typename T>
Vector<T>::Vector(const std::initializer_list<T>& lst) {
    size = lst.size();
    capacity = size;
    array = new T[size];
    std::copy(lst.begin(), lst.end(), array);
}

template <typename T>
Vector<T>& Vector<T>::operator=(const Vector<T>& vec) {

```

```

        if (this == &vec) {
            return *this;
        }

        this->resize(vec.size);
        for (size_t i = 0; i < this->size; ++i) {
            this->array[i].~T();
            new (this->array + i) T(vec[i]);
        }
        return *this;
    }

template <typename T>
T& Vector<T>::operator[](size_t index) {
    return array[index];
}

template <typename T>
const T& Vector<T>::operator[](size_t index) const {
    return array[index];
}

template <typename T>
Vector<T>::~~Vector() {
    for (size_t i = 0; i < size; ++i){
        array[i].~T();
    }
    delete[] reinterpret_cast<int8_t*>(array);
}

template <typename T>
size_t Vector<T>::get_capacity() const {
    return capacity;
}

template <typename T>
size_t Vector<T>::get_size() const {
    return size;
}

template <typename T>
const T& Vector<T>::front() const {
    if (size == 0) {
        throw std::range_error("vector is empty");
    }

    return array[0];
}

template <typename T>
T& Vector<T>::front() {
    if (size == 0) {
        throw std::range_error("vector is empty");
    }

    return array[0];
}

template <typename T>
const T& Vector<T>::back() const {
    if (size == 0) {
        throw std::range_error("vector is empty");
    }

    return array[size - 1];
}

```

```

template <typename T>
T& Vector<T>::back() {
    if (size == 0) {
        throw std::range_error("vector is empty");
    }

    return array[size - 1];
}

template <typename T>
void Vector<T>::reserve(size_t n) {
    if (n <= size) {
        return;
    }

    T* newArr = reinterpret_cast<T*>(new int8_t[n * sizeof(T)]);

    try {
        std::uninitialized_copy(array, array + size, newArr);
    } catch (...) {
        delete[] reinterpret_cast<int8_t*>(newArr);
        throw;
    }

    for (size_t i = 0; i < size; ++i) {
        array[i].~T();
    }

    delete[] reinterpret_cast<int8_t*>(array);
    array = newArr;
    capacity = n;
}

template <typename T>
void Vector<T>::resize(size_t n, const T& value) {
    if (n > capacity) {
        reserve(n);
    }

    for (size_t i = size; i < n; ++i) {
        new (array + i) T(value);
    }

    if (n < size) {
        size = n;
    }
}

template <typename T>
void Vector<T>::push_back(const T& data) {
    if (capacity == size) {
        reserve(2 * size);
    }
    new (array + size) T(data);
    ++size;
}

template <typename T>
template <typename... Args>
void Vector<T>::emplace_back(const Args& ...args) {
    if (capacity == size) {
        reserve(2 * capacity);
    }

    new (array + size) T(args...);
}

```

```

    ++size;
}

template <typename T>
void Vector<T>::pop_back() {
    --size;
    array[size].~T();
}

template <typename T>
T& Vector<T>::at(size_t index) {
    if (index >= size) {
        throw std::range_error("Array index out of range!");
    }
    return array[index];
}

template <typename T>
const T& Vector<T>::at(size_t index) const {
    if (index >= size) {
        throw std::range_error("Array index out of range!");
    }
    return array[index];
}

template <typename T>
void Vector<T>::clear() {
    size_t sz_copy = size;

    for (size_t i = 0; i < sz_copy; ++i) {
        array[i].~T();
        size--;
    }
}

template <typename T>
void Vector<T>::shrink_to_fit() {
    if (size == 0) {
        reserve(1);
        return;
    }

    reserve(size);
}

template <typename T>
bool Vector<T>::empty() const {
    return size;
}

template <typename T>
bool Vector<T>::operator==(const Vector<T>& vec) const {
    if (size != vec.size) {
        return false;
    }

    for (size_t i = 0; i < size; ++i) {
        if (vec.array[i] != array[i]) {
            return false;
        }
    }

    return true;
}

template <typename T>

```

```
bool Vector<T>::operator!=(const Vector<T>& vec) const {
    return !this->operator==(vec);
}
```

## Matrix.hpp

```
#ifndef MATRIX_HPP_INCLUDED
#define MATRIX_HPP_INCLUDED

#include "Vector.hpp"

class Matrix{
private:
    Vector<int> line_matrix;
    size_t matrix_lines;
    size_t matrix_columns;

public:
    Matrix();
    Matrix(size_t, size_t);

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

    void multiplication(const Vector<int>&);
    Vector<int> read_vector_from_string();
};

#include "../src/Matrix.cpp"

#endif
```

## Matrix.cpp

```
#include <iostream>
#include "../include/Matrix.hpp"

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);
}

Vector<int> Matrix::read_vector_from_string() {
    Vector<int> v;
    size_t size = 0;
    int value = 0;
    std::cout << "Enter size of vector-string: ";
    std::cin >> size;

    std::cout << "Enter " << size << " elements: ";
    for (size_t i = 0; i < size; ++i) {
        std::cin >> value;
        v.push_back(value);
    }

    return v;
}
```



```

void Matrix::multiplication(const Vector<int>& vector_string) {
    if (vector_string.get_size() != matrix_lines) {
        throw std::range_error("columns != lines");
    }

    Vector<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.get_size()) {
        size_t j = line_matrix[cur_line];

        if (j == 0) {
            ++cur_line;

            if (cur_line >= line_matrix.get_size() || line_matrix[cur_
line] == 0) {
                break;
            }

            j = line_matrix[cur_line];
        }

        size_t cur_column = cur_line + 1;
        size_t column_of_result_vector = line_matrix[cur_line] - 1;

        for (size_t column = cur_column; line_matrix[column] != 0; col
umn += 2) {
            int value = line_matrix[column + 1];
            result[line_matrix[column] - 1] += (vector_string[column_o
f_result_vector] * value);
            cur_line = column;
        }

        ++cur_line;
    }

    size_t non_zero_el = 0;
    std::cout << "Result: ";
    for (size_t i = 0; i != result.get_size(); ++i) {
        if (result[i] != 0) ++non_zero_el;
        std::cout << result[i] << " ";
    }
    std::cout << "\nCount of non-
zeros elements of the result: " << non_zero_el << "\n";
}

std::istream& operator>>(std::istream& is, Matrix& matrix) {
    matrix.line_matrix.pop_back();
    size_t line;
    size_t columns;
    int input_value;

    std::cout << "Enter count of lines, columns: ";

    is >> line >> columns;

    std::cout << '\n' << "Enter " << line * columns << " elements: ";

    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;
}

```

## Benchmark.cpp

```

#include <vector>
#include <chrono>
#include "../include/Vector.hpp"

void benchmark(){
    std::cout << "Initialization(100_000 elements):" << std::endl;

    std::chrono::steady_clock::time_point begin = std::chrono::steady_
clock::now();
    Vector<int> vec1(100000);
    std::chrono::steady_clock::time_point end = std::chrono::steady_cl
ock::now();

    std::cout << std::chrono::duration_cast<std::chrono::milliseconds>
(end - begin).count() << "[ms]" << std::endl;

    begin = std::chrono::steady_clock::now();
    std::vector<int> vec1(100000);
    end = std::chrono::steady_clock::now();

    std::cout << std::chrono::duration_cast<std::chrono::milliseconds>
(end - begin).count() << "[ms]" << std::endl;

    std::cout << std::endl;
    std::cout << "Push_back" << std::endl;

    begin = std::chrono::steady_clock::now();
    Vector<int> vec2;
    for (size_t i = 0; i < 1000000; ++i){
        vec2.push_back(i);
    }
    end = std::chrono::steady_clock::now();

    std::cout << std::chrono::duration_cast<std::chrono::milliseconds>
(end - begin).count() << "[ms]" << std::endl;

    begin = std::chrono::steady_clock::now();
    std::vector<int> vec2;

```

```

    for (size_t i = 0; i < 1000000; ++i){
        vec2.push_back(i);
    }
    end = std::chrono::steady_clock::now();

    std::cout << std::chrono::duration_cast<std::chrono::milliseconds>
(end - begin).count() << "[ms]" << std::endl;

    std::cout << std::endl;
    std::cout << "Pop_back:" << std::endl;

    begin = std::chrono::steady_clock::now();
    for (size_t i = 0; i < 1000000; ++i){
        vec2.pop_back();
    }
    end = std::chrono::steady_clock::now();

    std::cout << std::chrono::duration_cast<std::chrono::milliseconds>
(end - begin).count() << "[ms]" << std::endl;

    begin = std::chrono::steady_clock::now();
    for (size_t i = 0; i < 1000000; ++i){
        vec2.pop_back();
    }
    end = std::chrono::steady_clock::now();

    std::cout << std::chrono::duration_cast<std::chrono::milliseconds>
(end - begin).count() << "[ms]" << std::endl;

    std::cout << std::endl;
    std::cout << "Clear:" << std::endl;

    begin = std::chrono::steady_clock::now();
    vec2.clear();
    end = std::chrono::steady_clock::now();

    std::cout << std::chrono::duration_cast<std::chrono::milliseconds>
(end - begin).count() << "[ms]" << std::endl;

    begin = std::chrono::steady_clock::now();
    vec2.clear();
    end = std::chrono::steady_clock::now();

    std::cout << std::chrono::duration_cast<std::chrono::milliseconds>
(end - begin).count() << "[ms]" << std::endl;

    std::cout << std::endl;
    std::cout << "Shrink_to_fit:" << std::endl;

    begin = std::chrono::steady_clock::now();
    vec2.shrink_to_fit();
    end = std::chrono::steady_clock::now();

    std::cout << std::chrono::duration_cast<std::chrono::milliseconds>
(end - begin).count() << "[ms]" << std::endl;

    begin = std::chrono::steady_clock::now();
    vec2.shrink_to_fit();
    end = std::chrono::steady_clock::now();

    std::cout << std::chrono::duration_cast<std::chrono::milliseconds>
(end - begin).count() << "[ms]" << std::endl;

    std::cout << std::endl;
    std::cout << "Reserve(10000):" << std::endl;

```

```

begin = std::chrono::steady_clock::now();
for (size_t i = 1; i < 10000; ++i){
    vec2.reserve(i);
}
end = std::chrono::steady_clock::now();

std::cout << std::chrono::duration_cast<std::chrono::milliseconds>
(end - begin).count() << "[ms]" << std::endl;

begin = std::chrono::steady_clock::now();
for (size_t i = 1; i < 10000; ++i){
    vec2.reserve(i);
}
end = std::chrono::steady_clock::now();

std::cout << std::chrono::duration_cast<std::chrono::milliseconds>
(end - begin).count() << "[ms]" << std::endl;

std::cout << std::endl;
std::cout << "Resize:" << std::endl;

vec2.clear();
vec2.shrink_to_fit();
vec2.clear();
vec2.shrink_to_fit();

begin = std::chrono::steady_clock::now();
for (size_t i = 1; i < 10000; ++i){
    vec2.resize(i);
}
end = std::chrono::steady_clock::now();

std::cout << std::chrono::duration_cast<std::chrono::milliseconds>
(end - begin).count() << "[ms]" << std::endl;

begin = std::chrono::steady_clock::now();
for (size_t i = 1; i < 10000; ++i){
    vec2.resize(i);
}
end = std::chrono::steady_clock::now();

std::cout << std::chrono::duration_cast<std::chrono::milliseconds>
(end - begin).count() << "[ms]" << std::endl;

std::cout << std::endl;

Vector<int> vec3(1000000);
Vector<int> vec4(1000000);
std::vector<int> vec3(1000000);
std::vector<int> vec4(1000000);

std::cout << "==" << std::endl;

begin = std::chrono::steady_clock::now();
if (vec3 == vec4){
    end = std::chrono::steady_clock::now();
    std::cout << std::chrono::duration_cast<std::chrono::milliseco
nds>(end - begin).count() << "[ms]" << std::endl;
}

begin = std::chrono::steady_clock::now();
if (vec3 == vec4){
    end = std::chrono::steady_clock::now();
    std::cout << std::chrono::duration_cast<std::chrono::milliseco
nds>(end - begin).count() << "[ms]" << std::endl;
}

```

```

std::cout << std::endl;
std::cout << "!=" << std::endl;

begin = std::chrono::steady_clock::now();
if (!(vec3 != vec4)){
    end = std::chrono::steady_clock::now();
    std::cout << std::chrono::duration_cast<std::chrono::milliseco
nds>(end - begin).count() << "[ms]" << std::endl;
}

begin = std::chrono::steady_clock::now();
if (!(vec3 != vec4)){
    end = std::chrono::steady_clock::now();
    std::cout << std::chrono::duration_cast<std::chrono::milliseco
nds>(end - begin).count() << "[ms]" << std::endl;
}

std::cout << std::endl;
}

```

## Run.cpp

```

#include "../include/Vector.hpp"
#include "../include/Matrix.hpp"
#include "benchmark.cpp"
#include <iostream>

int main() {
    benchmark();

    Matrix matrix;

    std::cin >> matrix;
    matrix.multiplication(matrix.read_vector_from_string());
    return 0;
}

```

## Тесты производительности

Initialization(100\_000 elements):

0[ms]

0[ms]

Push\_back

14[ms]

80[ms]

Pop\_back:

3[ms]

43[ms]

Clear:

0[ms]

0[ms]

Shrink\_to\_fit:

0[ms]

0[ms]

Reserve(10000):

55[ms]

42[ms]

Resize:

171[ms]

0[ms]

==:

0[ms]

12[ms]

!=:

0[ms]

12[ms]

Мой вектор почти схож в скорости с stl вектором, я думаю, что это из-за похожих реализаций. Заметные различия наблюдаются в pop\_back, push\_back, resize.

Push\_back: я думаю, что мой push\_back работает быстрее из-за отсутствия реализации move семантики, а именно move конструктора и move конструктора перемещения. А также перегруженного метода push\_back для rvalue ссылки. Насколько мне известно, в реализации stl вектора используется такая конструкция как move\_if\_noexcept, которая при наличии move

конструктора помеченного поехсерт будет использовать мув, а если конструктор непомечен, то копировать объект. В моей реализации значение принимается по константной ссылке – значит копирования не будет. Также возможно мой вектор выигрывает из-за прямого вызова конструктора, деструктора, new и delete. Чего нет в stl реализации, там это реализуется через аллокаторы и надстройку над ними allocator\_traits, что скорее всего тоже замедляет работу.

Pop\_back: не совсем понимаю, почему мой вектор работает настолько быстрее stl вектора, видимо вызов деструктора через allocator\_traits работает довольно долго. Также на мой взгляд имеются и другие надстройки которые делают вектор более безопасным, но менее быстрым.

Resize: для stl вектора занял очень мало времени, я думаю что это как-то связано с мув семантикой и аллокаторами. Также возможно в стандартной реализации имеется какое-то условие, из-за которого не выполняется resize

## **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.