МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

Высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

оставь надежды всяк сие читающий (карин т.а.)

**Отчет по учебной практике**

**«**Структура хранения для матриц специального типа**»**

**Выполнил:**

студент группы 381706­1

Карин Тимофей Андреевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Проверил:**

ассистент каф. МОСТ ИИТММ

Лебедев Илья Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc1346448)

[3. Руководство пользователя 5](#_Toc1346449)

[4. Руководство программиста 6](#_Toc1346450)

[4.1. Описание структуры программы 6](#_Toc1346451)

[4.2 Описание структур данных 6](#_Toc1346452)

[4.3 Описание алгоритмов 8](#_Toc1346453)

[5. Заключение 10](#_Toc1346454)

[6. Список литературы 11](#_Toc1346455)

1. Введение

Матрица – математический объект, записываемый в виде прямоугольной таблицы элементов. Есть несколько способов представления матриц в памяти компьютера. Обычно она хранится в виде двумерного массива. Но в данной лабораторной работе будет рассмотрен отдельный тип матриц, а именно, треугольные матрицы. Эти матрицы отличаются от обычных тем, что треугольные матрицы квадратные и что выше (верхнетреугольные) или ниже (нижнетреугольные) главной диагонали они заполнены нулями. В связи с этим, можно сделать вывод, что представлять такие матрицы в виде двумерного массива не рационально (хоть и очень удобно).

В данной работе мы будем рассматривать нижнетреугольные матрицы, и, так как для нас самым дорогим ресурсом является память, мы будем хранить только те элементы, которые находятся ниже главной диагонали. Делать это мы будем с помощью векторов. Каждая строка матрицы представляет из себя вектор определённой длины (длина равна номеру строки, увеличенному на единицу).

2. Постановка задачи

Основной целью данной работы является создание библиотеки для работы с треугольными матрицами, то есть работая только с элементами, находящимися ниже главной диагонали. Для этого необходимо выполнить следующие задачи:

* Реализовать класс для работы с векторами;
* Реализовать класс для работы с матрицами, в котором матрица представлена как вектор векторов, размер которых увеличивается от единицы до размера матрицы.
* Проверить работоспособность данной библиотеки с помощью фреймворка для написания тестов Google Test.

3. Руководство пользователя

Пользователь не может работать с библиотекой напрямую, поэтому была написана программа, предоставляющая пользователю все основные функции работы с матрицами. Программа хранится в проекте *Vector* в файле *Matrix.cpp*.

Пользователю необходимо ввести размерность матриц, то есть количество строк и столбцов. Впоследствии, это значение нельзя изменить. Затем пользователю необходимо выбрать один из вариантов действий:

1. Ввести матрицы A и B;
2. Сложить матрицы A и B;
3. Вычесть из матрицы A матрицу B;
4. Умножить матрицу A на матрицу B;
5. Разделить матрицу A на матрицу B;
6. Поменять местами матрицы A и B;
7. Найти определитель матрицы A;
8. Найти обратную матрицу для матрицы A;
9. Вывести на экран матрицы A и B;
10. Поместить результат последней операции в матрицу A;
11. Выйти из программы.

Если в ходе работы пользователь попробует найти обратную матрицу для матрицы с нулевым определителем или разделить на такую матрицу, то программа завершит свою работу, а на экране пользователь увидит сообщение об ошибке.

4. Руководство программиста

4.1. Описание структуры программы

Программа состоит из 3 модулей:

* *VectorLib* – содежит библиотеку для работы с векторами и треугольными матрицами. Состоит из файлов *Vector.h* и *Matrix.h*, в которых хранятся шаблонные классы *TVector* и *TMatrix*.
* *Vector* – пример использования программы. Описан в главе 3.
* *VectorTest* – предназначен для тестирования библиотеки *VectorLib*. Состоит из файлов *VectorTest.cpp* и *MatrixTest.cpp*.

Кроме этих модулей в программе используется класс исключений *TException* и фреймворк Google Test.

4.2 Описание структур данных

Рассмотрим класс *TVector*. Класс шаблонный, *T –* шаблонный класс, присваиваемый элементам вектора. Рассмотрим поля и методы класса:

Со спецификатором доступа protected:

* *int size* – длина вектора (размерность);
* *T\* mas* – массив, в котором хранятся элементы вектора;

Со спецификатором доступа public:

* *TVector (int n = 0)* – конструктор;
* *TVector (const TVector <T> &A)* – конструктор копирования;
* *TVector<T>& operator=( const TVector<T> &A)* – оператор присваивания;
* *bool operator==( const TVector &A)* – проверка на равенство;
* *bool operator!=( const TVector &A)* – проверка на неравенство;
* *T& operator[](int i)* – оператор индексирования;
* *TVector<T> operator+( const TVector<T> &A)* – сложение двух векторов;
* *TVector<T> operator-( const TVector<T> &A)* – вычитание векторов;
* *T operator\*( const TVector<T> &A)* – скалярное умножение векторов;
* *virtual ~TVector()* – деструктор;
* *friend ostream& operator<< (ostream& ostr, const TVector<T1> &A)* – оператор вывода (*T1* то же самое, что и *T*);
* *friend istream& operator>> (istream& istr, TVector<T1> &A) –* оператор ввода (*T1* то же самое, что и *T*);

Рассмотрим шаблонный класс *TMatrix*. Наследуется от TVector<TVector<T>> со спецификатором доступа public. *T* – шаблонный класс, присваиваемый элементам матрицы. Поля отсутствуют, все методы имеют спецификатор доступа public. Рассмотрим их:

* *TMatrix (int n=0) –* конструктор;
* *TMatrix (TMatrix<T>& A) –* конструктор копирования;
* *TMatrix<T> operator+(const TMatrix<T>& A)* – сложение матриц;
* *TMatrix<T> operator-( const TMatrix<T>& A) –* вычитание матриц;
* *TMatrix<T> operator\*( const TMatrix<T>& A) –* умножение матриц;
* *TMatrix<T> operator/( const TMatrix<T>& A) –* деление матриц;
* *T GetDet() –* возвращает значение определителя;
* *TMatrix<T> GetInvers() –* возвращает обратную матрицу;
* *TMatrix(const TVector<TVector<T>> &A) –* конструктор приведения типа;
* *TMatrix<T>& operator=(const TMatrix<T>& A) –* оператор присваивания;
* *bool operator==( const TMatrix<T>& A) –* проверка на равенство;
* *bool operator!=( const TMatrix<T>& A) –* проверка на неравенство;
* *friend ostream& operator<<(ostream& ostr, TMatrix<T1> &A) -* оператор вывода (*T1* то же самое, что и *T*);
* *friend istream& operator>>(istream& istr, TMatrix<T1> &A) -* оператор вывода (*T1* то же самое, что и *T*);

4.3 Описание алгоритмов

Умножение векторов:

Проверяем размерность векторов. Если они не совпадают, выбрасываем исключение. Результатом умножения является сумма произведений элементов векторов с одинаковыми номерами.

Нахождение определителя матрицы:

Умножаем элементы, находящиеся на главной диагонали матрицы. Это произведение является определителем.

Нахождение обратной матрицы:

Проверяем, не равен ли определитель нулю. Если он ненулевой, то продолжаем. Копируем исходную матрицу, чтобы не изменить исходные данные. Также создаём ещё одну матрицу, она должна быть единичной. Далее работаем со строками матриц, при этом, все преобразования для обеих матриц идентичны. Несмотря на изменения будем называть эти матрицы основной и единичной соответственно.

Все строки матриц, делим на значение элемента, находящегося на главной диагонали в этой строке. Из основной матрицы получаемся матрица, на главной диагонали которой стоят единицы. Теперь надо поочерёдно, начиная с первой (с нулевой) строки, вычитать из всех нижестоящих строк данную строку, умноженную на значение элемента в исходной матрице, находящемся в той строке, из которой вычитаем, в столбце, равном номеру строки, которую вычитаем. После того, как мы проделаем такие действия с единичной матрицей (но значения элементов, на которые умножаются строки, берутся из основной), она станет равна обратной для исходной.

Умножение матриц:

Проверяем размерности. Создаём матрицу результата. Поочерёдно берём строки первой матрицы и поочерёдно умножаем на столбцы второй матрицы (умножаем как векторы). Результат умножения записываем в элемент матрицы результата, номер строки и столбца которого совпадают с номерами умноженных строки и столбца.

Сложение (вычитание матриц):

Проверяем размерность. Складываем (вычитаем) элементы с одинаковыми номерами строк и столбцов и записываем результат в отдельную матрицу на те же позиции. Эта матрица и будет результатом.

Деление матриц:

Умножаем первую матрицу (которую делим) на обратную матрицу для второй (на которую делим).

5. Заключение

В ходе проделанной работы были достигнуты следующие результаты:

* Реализован класс TVector, содержащий основные функции для работы с векторами;
* Написан класс TMatrix, предназначенный для работы с нижнетреугольными матрицами;
* Обработаны основные исключительные ситуации;
* Реализован пример использования программы;
* Написаны тесты, проверяющие все основные методы.

6. Список литературы

1. Ахо Альфред В, Хопкрофт Джон Э и Ульман Джеффри Д Структуры данных и алгоритмы [Книга]. - [б.м.] : Вильямс, 2003.
2. Лафоре Роберт Структуры данных и алгоритмы в Java [Книга]. - СПб : Питер, 2013. - 2 : стр. 704.
3. Павловская Т. А. C/C++ Программирование на языке высокого уровня [Книга]. - СПб : Питер, 2003.
4. Страуструп Бьерн Язык программирования C++ Бином, 2004.