Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт Информационных Технологий Математики и Механики

Отчёт по лабораторной работе

Лабораторная работа №2 Структуры хранения матриц специального вида

Выполнил:

Арутюнян Г.А.

Проверил:

Шагбазян Д.В.

Нижний Новгород

2020 г.

Содержание

[Введение 3](#__RefHeading___Toc270962758)

[Постановка задачи 4](#__RefHeading___Toc270962759)

[Руководство пользователя 5](#__RefHeading___Toc270962760)

[Руководство программиста 6](#__RefHeading___Toc270962761)

[Описание структур данных 6](#__RefHeading___Toc270962762)

[Описание алгоритмов 6](#__RefHeading___Toc270962763)

[Описание структуры программы 6](#__RefHeading___Toc270962764)

[Заключение 7](#__RefHeading___Toc270962765)

[Литература 8](#__RefHeading___Toc270962766)

[Приложения 9](#__RefHeading___Toc270962767)

[Приложение 1 9](#__RefHeading___Toc270962768)

[Приложение 2 9](#__RefHeading___Toc270962769)

# Введение

Понятие Матрица в европейской науке было введено в работах У. Гамильтона3 и А. Кэли4 в середине XIX века. Матричные обозначения широко распространены в современной математике и её приложениях. Матрица – полезный аппарат для исследования многих задач теоретической и прикладной математики. Так, одной из важнейших является задача нахождения решения систем линейных алгебраических уравнений. Следствием разнообразия областей применения матричного аппарата в современной науке является наличие в любом из больших математических программных комплексов (Mathcad, Mathematica, Derive, Mapple) подсистем, выполняющих операции над матрицами, а также существование специальных программных библиотек (ScalaPack, PlaPack), рассчитанных на обработку огромных (десятки и сотни тысяч строк) матриц, в том числе с использованием распределенных (параллельных) вычислений. Помимо матриц общего вида, для которых наиболее естественной и наиболее часто используемой представляется программная реализация в виде двумерного массива, в математических приложениях выделяются различные матрицы специальных видов (треугольные, диагональные, …). Для таких матриц предпочтительно создание собственных способов хранения и обработки, учитывающих специфику их структуры, и потому более эффективных. Изучению некоторых из них посвящена данная работа.

# Постановка задачи

В рамках лабораторной работы ставится задача создания программных средств, поддерживающих эффективное хранение матриц специального вида (верхнетреугольных) и выполнение основных операций над ними:

* сложение/вычитание;
* умножение;
* копирование;
* сравнение.

Программные средства должны содержать:

* класс Вектор (на шаблонах);
* класс Матрица (на шаблонах);
* тестовое приложение, позволяющее задавать матрицы и осуществлять основные операции над ними.

**Условия и ограничения**

Сделаем следующие основные допущения:

* Условимся рассматривать в дальнейшем верхнетреугольные квадратные матрицы, состоящие из элементов произвольного типа.
* Будем считать размер матрицы конечным числом, не превышающим 231.

# Руководство пользователя

Для начала работы с программой:

Запустите собранный проект (рекомендуется использовать Microsoft Visual Studio 2017, т.к. эта среда является средой разработки данной программы) и запустите файл Test\_Of\_Use.cpp. Далее опытным путём проверьте принцип работы программы, следуя инструкциям.

Так как матрица может быть представлена с помощью вектора векторов, она реализуется с помощью создания класса, наследованного от класса, реализующего вектор. (TVector.h, TVector.cpp, TMatrix.h, TMatrix.cpp)

Корректность работы каждого из классов(а также всей программы) проверяется с помощью соответствующих тестирующих файлов. (vector\_test.cpp, matrix\_test.cpp, main\_test.cpp)

Все проверки осуществляются с помощью Google-тестов, доступ к которым осуществляется с помощью файлов gtest.h, gtest-all.cc .

# Руководство программиста

## Описание структуры программы

Программа состоит из 5 значащих модулей и двух обязательных сопровождающих файлов(CMakeLists.txt и .travis.yml), позволяющих производить сборку всего проекта и его тестирование.

1. Модуль, содержащий google-тесты.

Состоит из gtest.h, gtest-all.cc, CMakeLists.txt

1. Модуль include

Содержит объявления классов в файлах TVector.h, TMatrix.h

1. Модуль src

Содержит реализации классов в файлах TVector.cpp, TMatrix.cpp (Класс TVector содержит методы: вычисление длины; сравнение; прибавление/вычитание скаляра; умножение на скаляр; сложение/вычитание векторов; скалярное произведение векторов; создание копии. Класс TMatrix содержит методы: сравнение;сложение/вычитание матриц; умножение матриц.), а также содержит необходимый для сборки сопутствующий файл CMakeLists.txt

1. Модуль samples

Содержит файл для проверки корректности работы программы пользователем Test\_Of\_Use.cpp, а также содержит необходимый для сборки сопутствующий файл CMakeLists.txt

1. Модуль test

Содержит файлы для тестирования методов классов и для тестирования всей программы в целом, используя google-тесты,(main\_test.cpp, matrix\_test.cpp, vector\_test.cpp) а также содержит необходимый для сборки сопутствующий файл CMakeLists.txt

## Описание структур данных

## Структура данных Вектора V = (v1, v2, …, vn) есть

## Sv = (Mv pv), где

## Mv = {v1,v2, …, vn} – базисное множество,

Pa= и, если j=i+1; л, если j≠i+1 - отношение следования

1. Структура данных Матрицы A = (aij), где i = 1..n; j = 1..n есть

Sa = (Ma, p1a, p2a), где

Ma = {a11, a12, …, ann-1,ann} – базисное множество,

P1a= и, если j=i+1; л, если j≠i+1 (k=1..n) - отношение следования

P2a= и, если j=i+1; л, если j≠i+1 (k=1..n) - отношение следования

## При определении Матрицы через Вектор (A = (vi), где vi – вектор из n элементов) структура данных примет вид:

## S1a = (M1a, p3a), где

## M1a = {v1, v2, …, vn} – базисное множество,

P3a= и, если j=i+1; л, если j≠i+1 - отношения следования

## Наконец, для верхнетреугольной матрицы имеет смысл задать структуру данных таким образом, чтобы исключить хранение нулевых элементов. Определение матрицы через вектор позволяет сделать это наилучшим образом:

## S2a = (M2a, p4a), где

## M2a = {v1, v2, …, vn} – базисное множество, где vi есть вектор из i элементов.

P4a= и, если j=i+1; л, если j≠i+1 - отношения следования

## Очевидно сходство в задании структуры данных Вектор, как набора элементов, связанных отношением следования, и структуры данных Матрица, как набора элементов-векторов, связанных отношением следования. Этот факт позволяет единообразно организовать алгоритмы обработки векторов и матриц, а, следовательно, использовать при разработке требуемых классов механизм наследования.

## Описание алгоритмов

TVector.h :

# #pragma once

# #ifndef TVector\_H

# #define TVector\_H

# #include <iostream>

# using namespace std;

# const int MAX\_VECTOR\_SIZE = 100000000;

# template <class Type>

# class TVector

# {

# protected:

# Type \*pVec;

# int Size;

# int StartIndex;

# public:

# TVector(int s = 10, int si = 0);

# TVector(const TVector &v);

# int GetSize() { return Size; };

# int GetStartIndex() { return StartIndex; };

# Type& GetValue(int pos);

# Type& operator[](int pos);

# bool operator==(const TVector &v);

# bool operator!=(const TVector &v);

# TVector& operator=(const TVector &v);

# TVector operator+(const Type &val);

# TVector operator-(const Type &val);

# TVector operator\*(const Type &val);

# TVector operator+(const TVector &v);

# TVector operator-(const TVector &v);

# Type operator\*(const TVector &v);

# friend istream& operator>>(istream &in, TVector &v)

# {

# for (int i = 0; i < v.Size; i++)

# in >> v.pVec[i];

# return in;

# }

# friend ostream& operator<<(ostream &out, TVector &v)

# {

# for (int i = 0; i < v.Size; i++)

# out << v.pVec[i] << " ";

# return out;

# }

# };

# template <class Type>

# TVector<Type>::TVector(int s, int si)

# {

# //if (s > MAX\_VECTOR\_SIZE || s<0 || si>s || si < 0)

# //{ throw out\_of\_range("Given size(s) must be less than 100000000 and greater than 0"); }

# Size = s;

# StartIndex = si;

# pVec = new Type[Size];

# for (int i = 0; i < Size; i++)

# pVec[i] = 0;

# }

# 

# template <class Type>

# TVector<Type>::TVector(const TVector&v)

# {

# Size = v.Size;

# StartIndex = v.StartIndex;

# pVec = new Type[Size];

# for (int i = 0; i < Size; i++)

# pVec[i] = v.pVec[i];

# }

# template<class Type>

# Type& TVector<Type>::GetValue(int pos)

# {

# //if (pos > Size || pos < StartIndex) throw out\_of\_range("You cannot reach this element. No such element. Out of range.");

# return pVec[pos - StartIndex];

# }

# template<class Type>

# Type& TVector<Type>::operator[](int pos)

# {

# return pVec[pos - StartIndex];

# }

# template<class Type>

# bool TVector<Type>::operator==(TVector const& v)

# {

# if (Size != v.Size || StartIndex != v.StartIndex)

# return false;

# for (int i = 0; i < v.Size; i++)

# {

# if (v.pVec[i] != pVec[i])

# return false;

# }

# return true;

# }

# template<class Type>

# bool TVector<Type>::operator!=(TVector const& v)

# {

# if (Size != v.Size || StartIndex != v.StartIndex)

# return true;

# for (int i = 0; i < v.Size; i++)

# {

# if (v.pVec[i] != pVec[i])

# return true;

# }

# return false;

# }

# template<class Type>

# TVector<Type>& TVector<Type>::operator=(const TVector &v)

# {

# if (this != &v)

# {

# if (Size != v.Size)

# {

# delete[] pVec;

# pVec = new Type[v.Size];

# }

# Size = v.Size;

# StartIndex = v.StartIndex;

# for (int i = 0; i < Size; i++)

# {

# pVec[i] = v.pVec[i];

# }

# }

# return \*this;

# }

# template<class Type>

# TVector<Type> TVector<Type>::operator+(Type const& val)

# {

# TVector<Type> tmp(\*this);

# for (int i = 0; i < this->Size; i++)

# tmp.pVec[i] += val;

# return tmp;

# }

# template<class Type>

# TVector<Type> TVector<Type>::operator-(Type const& val)

# {

# TVector<Type> tmp(\*this);

# for (int i = 0; i < this->Size; i++)

# tmp.pVec[i] -= val;

# return tmp;

# }

# template<class Type>

# TVector<Type> TVector<Type>::operator\*(const Type &val)

# {

# TVector<Type> tmp(\*this);

# for (int i = 0; i < this->Size; i++)

# tmp.pVec[i] \*= val;

# return tmp;

# }

# template<class Type>

# TVector<Type> TVector<Type>:: operator+(const TVector &v)

# {

# TVector<Type> tmp(\*this);

# if (v.Size != tmp.Size || v.StartIndex != tmp.StartIndex) throw length\_error("Vectors' sizes must be equal");

# for (int i = 0; i < tmp.Size; i++)

# tmp.pVec[i] = tmp.pVec[i] + v.pVec[i];

# return tmp;

# }

# template<class Type>

# TVector<Type> TVector<Type>:: operator-(const TVector &v)

# {

# TVector<Type> tmp(\*this);

# if (v.Size != tmp.Size || v.StartIndex != tmp.StartIndex) throw length\_error("Vectors' sizes must be equal");

# for (int i = 0; i < tmp.Size; i++)

# tmp.pVec[i] = tmp.pVec[i] - v.pVec[i];

# return tmp;

# }

# template<class Type>

# Type TVector<Type>::operator\*(const TVector &v)

# {

# Type tmp = 0;

# if (v.Size != Size || v.StartIndex != StartIndex) throw length\_error("Vectors' sizes must be equal");

# for (int i = 0; i < Size; i++)

# tmp = tmp + (this->pVec[i] \* v.pVec[i]);

# return tmp;

# }

#endif

# TVector.cpp :

# #include "TVector.h"

# TMatrix.h :

# #pragma once

# #ifndef TMatrix\_H

# #define TMatrix\_H

# #include "TVector.h"

# #include <iostream>

# const int MAX\_MATRIX\_SIZE = 10000;

# template<class Type>

# class TMatrix : public TVector<TVector<Type> >

# {

# public:

# TMatrix(int s = 10);

# TMatrix(const TMatrix &mt);

# TMatrix(const TVector<TVector<Type> > &mt);

# bool operator==(const TMatrix &mt);

# TMatrix& operator=(const TMatrix &mt);

# TMatrix operator+(const TMatrix &mt);

# TMatrix operator-(const TMatrix &mt);

# TMatrix operator\*(const TMatrix &mt);

# friend istream& operator>>(istream& in, TMatrix &mt)

# {

# for (int i = 0; i < mt.Size; i++)

# in >> mt.pVec[i];

# return in;

# }

# friend ostream& operator<<(ostream& out, TMatrix &mt)

# {

# for (int i = 0; i < mt.Size; i++)

# out << mt.pVec[i] << "\n";

# return out;

# }

# };

# template<class Type>

# TMatrix<Type>::TMatrix(int s) : TVector<TVector<Type> >(s)

# {

# for (int i = 0; i < s; i++)

# this->pVec[i] = TVector<Type>(s - i, i);

# }

# template<class Type>

# TMatrix<Type>::TMatrix(const TMatrix<Type> &mt) :TVector<TVector<Type> >(mt) {}

# template<class Type>

# TMatrix<Type>::TMatrix(const TVector<TVector<Type> > &mt) : TVector<TVector<Type> >(mt)

# {

# for (int i = 0; i < this->GetSize(); i++)

# {

# if (this->pVec[i].GetSize() != this->GetSize() - i)

# throw std::out\_of\_range("This is not a triangle matrix");

# }

# }

# template<class Type>

# TMatrix<Type>& TMatrix<Type>:: operator=(const TMatrix &mt)

# {

# if (this != &mt)

# {

# if (this->Size != mt.Size)

# {

# delete[] this->pVec;

# this->pVec = new TVector<Type>[mt.Size];

# }

# this->Size = mt.Size;

# this->StartIndex = mt.StartIndex;

# for (int i = 0; i < this->Size; i++)

# {

# this->pVec[i] = mt.pVec[i];

# }

# }

# return \*this;

# }

# template<class Type>

# bool TMatrix<Type>::operator==(TMatrix const& mt)

# {

# if (this->Size != mt.Size || this->StartIndex != mt.StartIndex)

# return false;

# for (int i = 0; i < mt.Size; i++)

# {

# if (mt.pVec[i] != this->pVec[i])

# return false;

# }

# return true;

# }

# template<class Type>

# TMatrix<Type> TMatrix<Type>::operator+(const TMatrix &mt)

# {

# TMatrix tmp(\*this);

# //if (mt.Size != tmp.Size || mt.StartIndex != tmp.StartIndex) throw length\_error("Matrixes' sizes must be equal");

# for (int i = 0; i < tmp.Size; i++)

# tmp.pVec[i] = tmp.pVec[i] + mt.pVec[i];

# return tmp;

# }

# template<class Type>

# TMatrix<Type> TMatrix<Type>::operator-(const TMatrix &mt)

# {

# TMatrix tmp(\*this);

# //if (mt.Size != tmp.Size || mt.StartIndex != tmp.StartIndex) throw length\_error("Matrixes' sizes must be equal");

# for (int i = 0; i < tmp.Size; i++)

# tmp.pVec[i] = tmp.pVec[i] - mt.pVec[i];

# return tmp;

# }

# template<class Type>

# TMatrix<Type> TMatrix<Type>::operator\*(const TMatrix &mt)

# {

# 

# //if (mt.Size != this->Size || mt.StartIndex != this->StartIndex) throw length\_error("Matrixes' sizes must be equal");

# TMatrix<Type> tmp(this->Size);

# for (int i = 0; i < this->Size; i++)

# for(int j=0;j<this->Size/2;j++) //size/2

# { Type tp = Type();

# for (int k = i; k <= j; ++k)

# {

# tp = tp + (this->pVec[i][k] \* mt.pVec[k][j]);

# }

# tmp.pVec[i][j] = tp;

# }

# return tmp;

# }

# #endif

# TMatrix.cpp :

#include "TMatrix.h"

# Эксперименты

Test\_Of\_Use.cpp :

#include <iostream>

#include "TVector.h"

#include "TMatrix.h"

int main()

{

setlocale(LC\_ALL, "Russian");

int size;

cout << "Ââåäèòå ðàçìåð ìàòðèöû: ";

cin >> size;

TMatrix<int> a(5), b(5), c(5),d(size),e(size);

int i, j;

cout << "Òåñòèðîâàíèå ïðîãðàìì ïîääåðæêè ïðåäñòàâëåíèÿ òðåóãîëüíûõ ìàòðèö" << endl;

for (i = 0; i < 5; i++)

for (j = i; j < 5; j++)

{

d[i][j] = i \* 10 + j;

e[i][j] = (i \* 10 + j) \* 100;

}

cout << "Matrix d" << endl << c << endl;

d = d \* 2;

cout << "Matrix d = a \* 2" << endl << c << endl;

cout << "Matrix e" << endl << e << endl;

cout << "Òåñòèðîâàíèå ïðîãðàìì ïîääåðæêè ïðåäñòàâëåíèÿ òðåóãîëüíûõ ìàòðèö" << endl;

for (i = 0; i < 5; i++)

for (j = i; j < 5; j++)

{

a[i][j] = i \* 10 + j;

b[i][j] = (i \* 10 + j) \* 100;

}

c = a + b;

cout << "Matrix a = " << endl << a << endl;

cout << "Matrix b = " << endl << b << endl;

cout << "Matrix c = a + b" << endl << c << endl;

c = b-a;

cout << "Matrix c = b - a" << endl << c << endl;

c = a \* b;

cout << "Matrix c = a \* b" << endl << c << endl;

return 0;}

# Заключение

Успешно создан корректный проект для представления матриц треугольного вида с помощью векторов векторов и работы с ними. Проект собран с помощью CMake и тестируется при поддержке travis CI.

# Литература

1. ЛАБОРАТОРНЫЙ ПРАКТИКУМ Учебно-методическое пособи

# Приложения