Государственный Университет Молдовы

Факультет Математики и Информатики

Кафедра Информатики

**Лабораторная работа №4**

по курсу «Алгоритмы и Структуры Данных»

Методы доступа к элементам массива.

Выполнил студент группы I2102:

**Король Владимир**

Проверил преподаватель:

**Кроитор Михаил**

***Метод прямого доступа к элементам массива.***

***Ускоренный метод с помощью определяющих векторов.***

***Метод доступа посредством вектора Айлиффа.***

***Сравнить методы доступа по времени выполнения.***

***Сравнить методы доступа по используемой памяти.***

l1 = 1, h1 = 4,   
l2 = -8, h2 = -6,   
l3 = 0, h3 = 2,   
l4 = -2, h4 = 0;  
  
Базовым классом для всех классов, реализующих доступ к многомерному массиву, будет класс **Matrix4Base**. Конструктор будет принимать начало и конец индексов каждого измерения, а также функции заполнения массива по текущему индексу. Также класс содержит указатель на одномерный массив всех элементов.

Остальные классы, работающие с массивом и реализующие задание данной лабораторной работы, наследуют данный базовый класс, а именно:  
  
• **Matrix4C** – прямой доступ, распределение по колонкам

• **Matrix4L** – прямой доступ, распределение по строкам

• **Matrix4CDV** – доступ через определяющий вектор, распределение по колонкам

• **Matrix4LDV** – доступ через определяющий вектор, распределение по строкам

• **Matrix4CIV** – доступ через вектор Айлиффа, распределение по колонкам

• **Matrix4LIV** – доступ через вектор Айлиффа, распределение по строкам

Таким образом, функция получения элемента по индексам, будет иметь следующий вид:

this->D4 = 1;

this->D3 = (this->h4 - this->l4 + 1) \* this->D4;

this->D2 = (this->h3 - this->l3 + 1) \* this->D3;

this->D1 = (this->h2 - this->l2 + 1) \* this->D2;

И для распределения по колонкам:

this->D1 = 1;

this->D2 = (this->h1 - this->l1 + 1) \* this->D1;

this->D3 = (this->h2 - this->l2 + 1) \* this->D2;

this->D4 = (this->h3 - this->l3 + 1) \* this->D3;

Получение элемента при использовании определяющего выглядит следующим образом:

virtual T& elem(int i1, int i2, int i3, int i4) override {

return Matrix4Base<T>::source[i1 \* d[8] + i2 \* d[9] + i3 \* d[10] + i4 \* d[11] - d[12]];}

Где последние элементы вектора d:

d[7] = l4;

d[8] = h4;

d[7] = Matrix4Base<T>::num; //num of elements

d[8] = Matrix4L<T>::D1;

d[9] = Matrix4L<T>::D2;

d[10] = Matrix4L<T>::D3;

d[11] = Matrix4L<T>::D4;

d[12] = l1 \* Matrix4L<T>::D1 + l2 \* Matrix4L<T>::D2 + l3 \* Matrix4L<T>::D3 + l4 \* Matrix4L<T>::D4; //sum(ljxDj)

При использовании векторов Айлиффа, следует определить вектора, содержащие элементы массива:

T\*\*\*\* vect;

public:

Matrix4CIV(int l1, int h1, int l2, int h2, int l3, int h3, int l4, int h4, randomize<T> randomizer)

: Matrix4C<T>(l1, h1, l2, h2, l3, h3, l4, h4, randomizer) {

int i2, i3, i4, d, step;

d = -l1;

step = h1 - l1 + 1;

vect = new T\*\*\*[h4 - l4 + 1] - l4;

for (i4 = l4; i4 <= h4; i4++) {

\*(vect + i4) = new T\*\*[h3 - l3 + 1] - l3;

for (i3 = l3; i3 <= h3; i3++) {

\*(\*(vect + i4) + i3) = new T\*[h2 - l2 + 1] - l2;

for (i2 = l2; i2 <= h2; i2++, d += step) {

\*(\*(\*(vect + i4) + i3) + i2) = Matrix4Base<T>::source + d;

}

}

}

}

virtual T& elem(int i1, int i2, int i3, int i4) override {

return \*(\*(\*(\*(vect + i4) + i3) + i2) + i1);

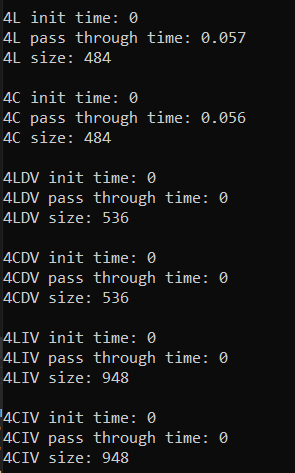
}

Аналогично распределение по строкам, только в данном случае инициализация и получение доступа будет происходить в обратном порядке.

**Вывод на основе сравнения результатов (результаты в конце документа):**Согласно полученным результатам, прямой доступ к массиву занимает больше времени в пробеге всего массива, но в тоже время, выделяемая память для хранения данных сравнительно меньше, чем в двух других случаях.

Доступ через определяющие вектора работает быстрее прямого и занимает ненамного больше памяти.

Доступ через вектора Айлиффа также быстрый, но занимает в два раза больше памяти.

****