Министерство Образования, Культуры и Исследований

**Молдавский Государственный Университет**

**Факультет Математики и Информатики**

**Департамент Информатики**

Algoritmi, Structuri de Date și Complexitate

Отчёт по лабораторной работе №4

Выполнил: Maryia Rotar IA2102

Проверил преподаватель: Mihail Croitor

Кишинев, 2023

**Задание:**

Задан многомерный массив/вектор B (дана размерность массива и интервалы изменения индексов). Создать класс с тремя методами, которые реализуют:

1. прямой доступ к элементам вектора,
2. доступ посредством векторов Айлиффа, и
3. метод определяющих векторов (представление в оперативной памяти производится по столбцам и строкам).

Вывести на экран время доступа к элементам массива и провести сравнения.

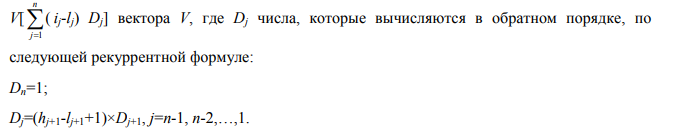
* Метод прямого доступа к элементам массива - 2 балла
* Ускоренный метод с помощью определяющих векторов - 3 балла
* Метод доступа посредством вектора Айлиффа - 3 балла
* Сравнить методы доступа по времени выполнения - 1 балл
* Сравнить методы доступа по используемой памяти - 1 балл

Матрица - структура данных, состоящая из конечного набора элементов одного и того же типа, расположенных в оперативной памяти так, что позиция каждого элемента матрицы однозначно определяется упорядоченным набором целых чисел (индексов). Индексы обеспечивают возможность прямого доступа к каждому элементу матрицы.

Вектор представляет собой упорядоченный конечный набор элементов одного и того же типа, размещаемых физически в памяти компьютера последовательно один за другим, без промежутков.

**Метод прямого доступа к элементам массива**

Прямой доступ к элементу матрицы, отображенной в векторе, вычислятся по следующей формуле:



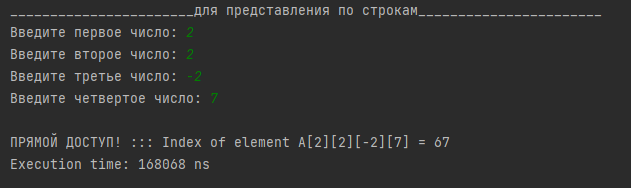


Рис.1 Результат работы вычисления по прямому доступу.

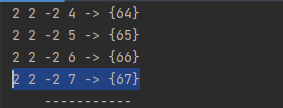


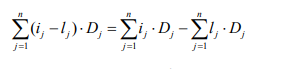
Рис.2 Проверка элемента в матрице (в примере элементы – числа, которые совпадают с индексами).

//прямой доступ к элементам для реализации "представление матриц по строкам"  
public int getElementByDirectWay(int i1, int i2, int i3, int i4){  
  
 int i\_j[] = {i1, i2, i3, i4};  
  
 int sum = 0;  
  
 for (int j = 0; j < n; j++){  
  
 //V[sum\_1\_n(i\_j - l\_j) × Dj]  
 sum += (i\_j[j] - Dimension.*getDim*(j+1).getStart()) \* d\_j[j];  
  
 }  
 return sum;  
}

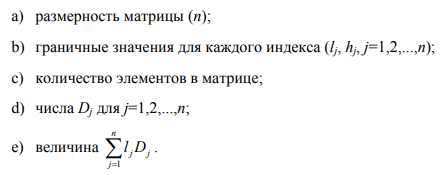
private void findD\_j() {  
  
 int D\_j = 1; //Dn = D4  
 for (int i = n - 1; i >= 0; i--) {  
  
 this.d\_j[i] = D\_j;  
 int D\_prev = D\_j;  
  
 //D(j+1), где j=n-1, n-2, ... 1;  
 //Dj=(h(j+1)-l(j+1)+1)×D(j+1)  
 D\_j = (Dimension.*getDim*(i + 1).getEnd() - Dimension.*getDim*(i + 1).getStart() + 1) \* D\_prev;  
  
 }  
}

**Ускоренный метод с помощью определяющих векторов**

Этот метод вычисляется по формуле:



При этом, для экономии времени выполнения следующие величины записываются в отдельный вектор:



public int getElementByDefiningVector(int i1, int i2, int i3, int i4){  
  
 int i\_j[] = {i1, i2, i3, i4};  
  
 //В определяющий вектор записываются следующие величины:  
 //a) размерность матрицы (n); // this.n  
 //b) граничные значения для каждого индекса (lj, hj, j=1,2,...,n); // enum Dimension  
 //c) количество элементов в матрице; // this.numberOfElements  
 //d) числа Dj для j=1,2,...,n; // this.d\_j[]  
 //e) sum\_1\_n ( l\_j \* D\_j); // this.sum\_lj\_mul\_dj  
  
 /\* все значения уже есть в этом классе; будем считать, что извлекаем их из опред-го вектора \*/  
  
 int sum2 = 0;  
 for (int j = 0; j < n; j++){  
 sum2 += i\_j[j] \* d\_j[j];  
 }  
  
 return sum2 - this.sum\_lj\_mul\_dj;  
}

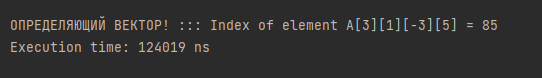


Рис.3 Результат работы определяющего вектора.

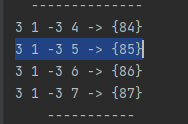
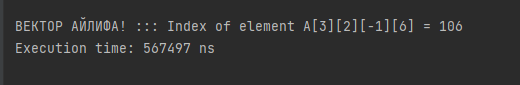


Рис.4 Проверка по матрице.

**Метод доступа посредством вектора Айлиффа**

Суть метода состоит в том, что наряду с каждой матрицей строится и сохраняется иерархия векторов, получившая название векторов Айлиф. С именем матрицы ассоциируется указатель с адресом первого вектора Айлифа смещенным в соответствии с нижним значением первого индекса в случае размещения элементов ―по строкам, или с нижним значением последнего индекса в случае размещения элементов ―по столбцам.



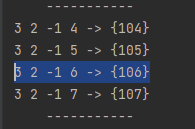


Рис.5 Результат работы вектора Айлифа.

private static List<List<List<List<Integer>>>> *vectorIliffe*;  
  
public static void makeIliffeVector() {  
 int filler = 0;  
 *vectorIliffe* = new ArrayList<>(*sizeDim1*);  
 for (int i = 0; i < *sizeDim1*; i++) {  
 List<List<List<Integer>>> level1 = new ArrayList<>(*sizeDim2*);  
 for (int j = 0; j < *sizeDim2*; j++) {  
 List<List<Integer>> level2 = new ArrayList<>(*sizeDim3*);  
 for (int k = 0; k < *sizeDim3*; k++) {  
 List<Integer> level3 = new ArrayList<>(*sizeDim4*);  
 for (int l = 0; l < *sizeDim4*; l++) {  
 level3.add(filler);  
 //System.out.println(" el: " + filler);  
 filler++;  
 }  
 level2.add(level3);  
 }  
 level1.add(level2);  
 }  
 *vectorIliffe*.add(level1);  
 }  
}

public static int getElementByIliffeVector(int i1, int i2, int i3, int i4){  
  
 if (*vectorIliffe*.size() > 0){  
  
 int ii = 0, jj=0, kk=0, ll=0;  
 for (int i = Dimension.*getDim*(1).getStart(); i < i1; i++, ii++){}  
 for (int j = Dimension.*getDim*(2).getStart(); j < i2; j++, jj++){}  
 for (int k = Dimension.*getDim*(3).getStart(); k < i3; k++, kk++){}  
 for (int l = Dimension.*getDim*(4).getStart(); l < i4; l++, ll++){}  
  
 return *vectorIliffe*.get(ii).get(jj).get(kk).get(ll);  
 }  
 return 0;  
}

Данный код сначала формирует матрицы Айлифа и далее просто получает элемент по ссылке со смещением.

**Выводы**

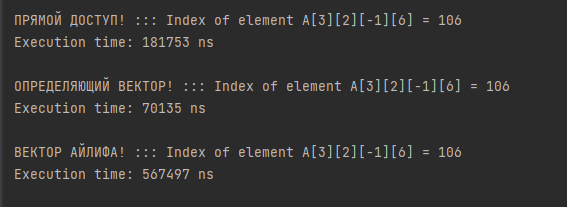


Рис.6 Сравнение времени работы всех методов. Видно, что наиболее быстрый – метод определяющего вектора.