Министерство образования Российской Федерации
Московский Государственный Технический
Университет им. Н.Э. Баумана

Отчет по лабораторной работе №2
По курсу «Анализ алгоритмов»

Тема: «Умножение матриц»

Студент: **Горохова И.Б.**
Группа: **ИУ7-51**

Преподаватель: **Волкова Л.Л.**

Москва, 2017

Содержание

Постановка задачи

В ходе лабораторной работы предстоит:

1. Изучить алгоритмы умножения матриц: стандартный и алгоритм Винограда
2. Улучшить алгоритм Винограда
3. Дать теоретическую оценку базового алгоритма умножения матриц, алгоритма Винограда и улучшенного алгоритма Винограда
4. Реализовать три алгоритма умножения матриц на одном из языков программирования
5. Сравнить алгоритмы умножения матриц

Описание алгоритмов

Базовый алгоритм умножения матриц

Для вычисления произведения двух матриц A и B каждая строка матрицы A умножается на каждый столбец матрицы B. Затем подсчитывается сумма таких произведений и записывается в соответствующую ячейку результирующей матрицы.

Пример умножения двух матриц:

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} * \begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} = \begin{bmatrix} aA + bD + cG & aB + bE + cH & aC + bF + cI \\ dA + eD + fG & dB + eE + fH & dC + eF + fI \end{bmatrix}$$

Листинг 1: Базовый алгоритм умножения матриц

```
public long BaseMuliplication(Matrix matrix2, Matrix
    resultMatrix){
  //...
  for (int i = 0; i < resultMatrix.m; i++)
    for (int j = 0; j < resultMatrix.n; j++)
      for (int k = 0; k < this.n; k++)
        resultMatrix[i][j] = resultMatrix[i][j] + this.
            matrix[i][k] * matrix2[k][j];
  //...
}
```

Входные данные: $matrix$ - первая матрица, $matrix2$ - вторая матрица
Выходные данные: $resultMatrix$ - результирующая матрица

Алгоритм Винограда

Алгоритм Винограда считается более эффективным благодаря сокращению количества операций умножения. Результат умножения двух матриц представляет собой скалярное произведение соответствующих строки и столбца. Можно заметить, что такое умножение позволяет выполнить заранее часть работы:
$U = (u_1, u_2, u_3, u_4)$ и $V = (v_1, v_2, v_3, v_4)$
$U * V = u_1 * v_1 + u_2 * v_2 + u_3 * v_3 + u_4 * v_4$
Это равенство можно переписать в виде
$U*V = (u_1+v_2)*(v_1+u_1)+(u_3+v_4)*(u_4+v_3)-u_1*u_2-u_3*u_4-v_1*v_2-v_3*v_4$
При этом умножения $u_1*u_2, u_3*u_4, v_1*v_2, v_3*v_4$ можно рассчитать заранее.

Однако с точки зрения реализации, под массивы строковых и столбцовых коэффициентов требуется дополнительная память. Так же, в случае нечетного количества столбцов первой матрицы (строк второй матрицы) требуются дополнительные вычисления.

Листинг 2: Алгоритм Винограда умножения матриц

```
public long VinogradMultiplication(Matrix matrix2, Matrix
    resultMatrix){
    //...
  for (int i = 0; i < resultMatrix.m; i++)
  {
    rowFactor[i] = this.matrix[i][0] * this.matrix[i][1];
    for (int j = 1; j < this.n/2; j++)
      rowFactor[i] = rowFactor[i] + this.matrix[i][2*j]*
          this.matrix[i][2*j+1];
  }

  for (int i = 0; i < resultMatrix.n; i++)
  {
    columnFactor[i] = matrix_2[0][i] * matrix_2[1][i];
    for (int j = 1; j < this.n/2; j++)
      columnFactor[i] = columnFactor[i] + matrix2[2*j][i] *
          matrix2[2*j+1][i];
  }

  for (int i = 0; i < resultMatrix.m; i++)
    for (int j = 0; j < resultMatrix.n; j++)
```

```
    {
      res[i][j] = - rowFactor[i] - columnFactor[j];
      for (int k = 0; k < this.n/2; k++)
        resultMatrix[i][j] = resultMatrix[i][j] + (this.
            matrix[i][2*k] + matrix2[2*k+1][j])*(this.matrix
            [i][2*k+1] + matrix2[2*k][j]);
    }

  if (this.n % 2 == 1)
    for (int i = 0; i < resultMatrix.m; i++)
      for (int j = 0; j < resultMatrix.n; j++)
        resultMatrix[i][j] = resultMatrix[i][j] + this.
            matrix[i][this.n-1] * matrix2[this.n-1][j];
  //...
}
```

Входные данные: $matrix$ - первая матрица, $matrix2$ - вторая матрица
Выходные данные: $resultMatrix$ - результирующая матрица

Улучшенный алгоритм Винограда

Улучшения:

- Замена операции = ...+ на + =
- Добавление буфера $buffer$ для уменьшения количества обращений к результирующей матрице
- Замена $buffer = -rowFactor[i] - columnFactor[j]$; на $buffer- = rowFactor[i] + columnFactor[j]$; для уменьшения количества операций с трех (=, -, -) до двух (-=, +)
- Избавление от циклов, в которых добавлялись члены в случае нечетной общей размерности. Замена их (в основном цикле) на тернарное выражение, которым инициализируется буфер при $n\%2 = 1$: $buffer = (flag?this.matrix[i][this.n-1] * matrix_2[this.n-1][j] : 0)$, где $booleanflag = this.n\%2 == 1$;

Листинг 3: Улучшенный алгоритм Винограда умножения матриц

```
public long VinogradImpMultiplication(Matrix matrix2,
    Matrix resultMatrix){
    //...
  int d = this.n/2;
  boolean flag = this.n % 2 == 1;
```

```
  for (int i = 0; i < resultMatrix.m; i++)
  {
    rowFactor[i] = this.matrix[i][0] * this.matrix[i][1];
    for (int j = 1; j < d; j++)
      rowFactor[i] += this.matrix[i][2*j]*this.matrix[i][2*j+1];
  }

  for (int i = 0; i < resultMatrix.n; i++)
  {
    columnFactor[i] = matrix_2[0][i] * matrix2[1][i];
    for (int j = 1; j < d; j++)
      columnFactor[i] += matrix2[2*j][i] * matrix2[2*j+1][i];
  }

  for (int i = 0; i < resultMatrix.m; i++)
    for (int j = 0; j < resultMatrix.n; j++)
    {
      buffer = (flag ? this.matrix[i][this.n-1] * matrix2[this.n-1][j] : 0);
      buffer -= rowFactor[i] + columnFactor[j];
      for (int k = 0; k < d; k++)
        buffer += (this.matrix[i][2*k] + matrix2[2*k+1][j])*(this.matrix[i][2*k+1] + matrix2[2*k][j]);
      res[i][j] = buffer;
    }
  //...
}
```

Входные данные: $matrix$ - первая матрица, $matrix2$ - вторая матрица
Выходные данные: $resultMatrix$ - результирующая матрица

Теоретическая оценка

Базовый алгоритм умножения матриц

$f_a = 2 + m * (2 + 2 + n * (2 + 2 + N * (2 + 11)))$

Трудоёмкость алгоритма: **13Nmn + 4mn + 4m + 2**

Алгоритм Винограда

$f_a = 2 * [2 + m * (2 + 5 + 2 + 2 + (N/2 - 1) * (3 + 6 + 6))] +$
$+ [2 + m * (2 + 2 + n * (2 + 4 + 3 + 2 + N/2 * (3 + 12 + 11)))] +$
$+ [2 + 2 + m * (2 + 2 + n * (2 + 8 + 5))]$

Последнее слагаемое входит в трудоёмкость худшего случая (когда общая размерность нечётная).
В лучшем случае последнее слагаемое = 2 (операции проверки на четность)

Трудоёмкость алгоритма:
Худший случай: **12Nmn + 41mn + 6m + 10**
Лудший случай: **12Nmn + 26mn - 2m +8**

Улучшенный алгоритм Винограда

$f_a = 2 * [2 + m * (2 + 5 + 2 + 2 + (N/2 - 1) * (2 + 5 + 5))] +$
$+ [2 + m * (2 + 2 + n * (2 + \mathbf{[4 + 3 + 2]}/\mathbf{[2]} + 2 + 2 + 2 +$
$+ N/2 * (2 + 8 + 10) + 3))]$

Выделенная курсивом часть представляет собой слагаемое для [худшего]/[лучшего] случая

Трудоёмкость алгоритма:
Худший случай: **10Nmn + 32mn + 2m + 6**
Лудший случай: **10Nmn + 25mn + 2m +6**

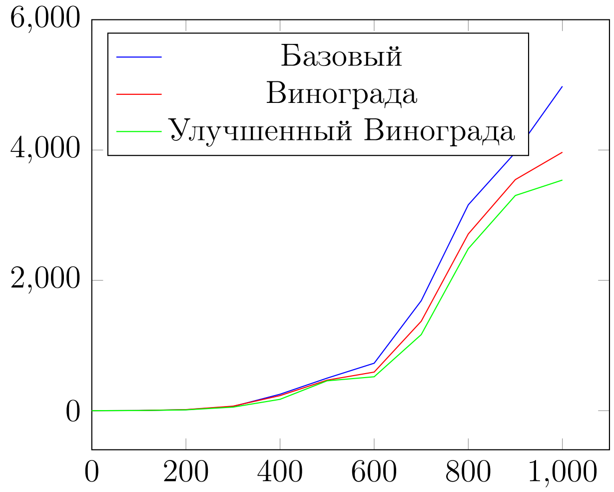
Эксперимент

В качестве эксперимента произведены по десять замеров времени работы каждого из трех алгоритмов умножения для квадратных матриц размерностями 100*100, 200*200 ... 1000*1000 и 101*101, 201*201 ... 1001*1001 Среднее значение времени работы алгоритмов в миллисекундах приведено в таблицах и на графиках.

*Время замерялось с помощью фунуции $publicstaticlongcurrentTimeMillis()$ из $System$

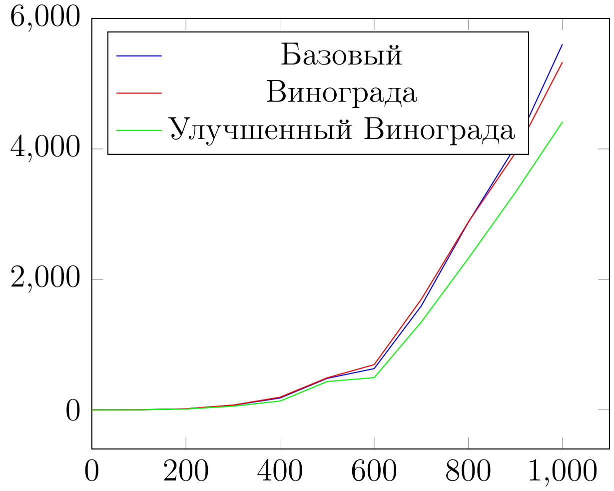
Матрицы чётной размерности

Размерность матрицы	**Базовый алгоритм**	**Алгоритм Винограда**	**Улучшенный алг. Винограда**
100*100	2	2	2
200*200	14	16	12
300*300	64	69	54
400*400	230	231	174
500*500	481	468	458
600*600	630	592	520
700*700	1287	1372	1167
800*800	2959	2712	2486
900*900	3962	3546	3302
1000*1000	4978	3966	3339



Матрицы нечётной размерности

Размерность матрицы	Базовый алгоритм	Алгоритм Винограда	Улучшенный алг. Винограда
101*101	1	1	1
201*201	15	17	13
301*301	70	73	56
401*401	181	194	135
501*501	482	491	434
601*601	632	693	494
701*701	1592	1694	1346
801*801	2875	2879	2322
901*901	4037	3938	3334
1001*1001	5605	5331	4413



Выводы из экспериментов

В результате экспериментов были подтверждены результаты теоретической оценки алгоритмов. Улучшенный алгоритм Винограда быстрее стандартного примерно в 1.22 раза для матриц чётных размерностей, и в 1.2 раза для матриц нечётных размерностей. Базовый алгоритм умножения матриц медленнее улучшенного алгоритма Винограда примерно в 1.36 раза для матриц чётных размерностей, и в 1.29 раза для матриц нечётных размерностей.

Заключение

В ходе лабораторной работы я изучила алгоритмы умножения матриц: стандартный и алгоритм Винограда, улучшила алгоритм Винограда, дала теоретическую оценку базового алгоритма умножения матриц, алгоритма Винограда и улучшенного алгоритма Винограда, реализовала три алгоритма умножения матриц на языке программирования и сравнила алгоритмы умножения матриц.