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: **Медведев А.В**
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. Сделать теоретическую оценку алгоритмов умножения матриц
4. Реализовать три алгоритма умножения матриц
5. Сравнить время работы алгоритмов 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 сумма данных произведений записывается в соответствующую ячейку результирующей матрицы.

Листинг 1: Базовый алгоритм умножения матриц

```
public static double[,] BaseMultiplication(double[,]
    matrix1, double[,] matrix2)
{
    int resRow = matrix1.GetLength(0);
    int resCol = matrix2.GetLength(1);
    int kk = matrix2.GetLength(0);
    double[,] resultmatrix=new double[resRow,resCol];

    for (int i = 0; i < resRow; i++)
    {
        for (int j = 0; j < resCol; j++)
        {
            for (int k = 0; k < kk; k++)
            {
                resultmatrix[i, j] = resultmatrix[i, j] +
                    matrix1[i, k] * matrix2[k, j];
            }
```

```
16          }
17      }
18
19      return resultmatrix;
20  }
```

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:
$U = (u_1, u_2, u_3, u_4)$
$V = (v_1, v_2, v_3, v_4)$
$U * V = u_1 * v_1 + u_2 * v_2 + u_3 * v_3 + u_4 * v_4$
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 под массивы данных коэффициентов требуется дополнительная память. В случае нечетного количества столбцов первой матрицы требуются дополнительные вычисления.

Листинг 2: Алгоритм Винограда

```
1   public static double[,] VinogradMultiplication(double[,]
        matrix1, double[,] matrix2)
2   {
3       int resRow = matrix1.GetLength(0);
4       int resCol = matrix2.GetLength(1);
5       int kk = matrix2.GetLength(0);
6       double[,] resultmatrix=new double[resRow,resCol];
7
8       double[] rowFactor=new double[resRow];
9       double[] colFactor=new double[resCol];
10
11      for (int i = 0; i <resRow; i++)
12      {
13          rowFactor[i] = matrix1[i, 0] * matrix1[i, 1];
14          for (int j = 1; j < kk/2; j++)
15          {
16              rowFactor[i] = rowFactor[i] + matrix1[i, 2 * j]
                    * matrix1[i, 2 * j + 1];
```

```
17          }
18      }
19      for (int i = 0; i < resCol; i++)
20      {
21          colFactor[i] = matrix2[0, i] * matrix2[1, i];
22          for (int j = 1; j < kk/2; j++)
23          {
24              colFactor[i] = colFactor[i] + matrix2[2 * j, i]
                    * matrix2[2 * j + 1, i];
25          }
26      }
27      for (int i = 0; i < resRow; i++)
28      {
29          for (int j = 0; j < resCol; j++)
30          {
31              resultmatrix[i, j] = -rowFactor[i] - colFactor[
                    j];
32              for (int k = 0; k < kk/2; k++)
33              {
34                  resultmatrix[i, j] = resultmatrix[i, j] +
35                                       (matrix1[i, 2 * k] +
                                            matrix2[2 * k + 1,
                                            j]) *
36                                       (matrix1[i, 2 * k + 1]
                                            + matrix2[2 * k, j
                                            ]);
37              }
38          }
39      }
40      if (kk%2==1)
41      {
42          for (int i = 0; i < resRow; i++)
43          {
44              for (int j = 0; j < resCol; j++)
45              {
46                  resultmatrix[i, j] = resultmatrix[i, j] +
                        matrix1[i, kk - 1] * matrix2[kk - 1, j];
47              }
48          }
49      }
50      return resultmatrix;
51  }
```

Улучшенный алгоритм Винограда

Улучшения:

- Замена = ...+ 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 (=, -, -) на (-=, +)
- Избавление от циклов для случая нечетной общей размерности. Замена их (в основном цикле)
  if $n\%2 = 1$:
  $buffer = (flag?matrix1[i, n-1] * matrix2[n-1, j] : 0);$

Листинг 3: Улучшенный алгоритм Винограда

```
public static double[,] BetterVinogradMultiplication(double
    [,] matrix1, double[,] matrix2)
{
    int resRow = matrix1.GetLength(0);//m
    int resCol = matrix2.GetLength(1);//n

    int n = matrix2.GetLength(0);
    int kk = matrix2.GetLength(0)/2;
    bool flag = n % 2 == 1;

    double[,] resultmatrix=new double[resRow,resCol];
    double[] rowFactor=new double[resRow];
    double[] colFactor=new double[resCol];

    for (int i = 0; i <resRow; i++)
    {
        rowFactor[i] = matrix1[i, 0] * matrix1[i, 1];
        for (int j = 1; j < kk; j++)
        {
            rowFactor[i] += matrix1[i, 2 * j] * matrix1[i,
                2 * j + 1];
        }
    }


```

```
24      for (int i = 0; i < resCol; i++)
25      {
26          colFactor[i] = matrix2[0, i] * matrix2[1, i];
27          for (int j = 1; j < kk; j++)
28          {
29              colFactor[i] += matrix2[2 * j, i] * matrix2[2 *
                    j + 1, i];
30          }
31      }
32
33      double buffer;
34      for (int i = 0; i < resRow; i++)
35      {
36          for (int j = 0; j < resCol; j++)
37          {
38              buffer = (flag ? matrix1[i, n - 1] * matrix2[n
                    - 1, j] : 0);
39              buffer -= rowFactor[i] + colFactor[j];
40              for (int k = 0; k < kk; k++)
41              {
42                  buffer +=(matrix1[i, 2 * k] + matrix2[2 * k
                        + 1, j]) *
43                                          (matrix1[i, 2 * k + 1]
                                                + matrix2[2 * k, j
                                            ]);
44
45              }
46              resultmatrix[i, j] = buffer;
47          }
48      }
49      return resultmatrix;
50
51  }
```

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

$f_a = 2 + m * (2 + 7 + 3 + (N/21)(12 + 3)) +$
$+ 2 + n * (2 + 7 + 3 + (N/21)(12 + 3)) +$
$+ 2 + m * (2 + 2 + n * (2 + 7 + 3 + N/2 * (3 + 22))) +$
$+ 2 + 2 + m * (2 + 2 + n * (2 + 13))$

Последнее слагаемое входит в трудоёмкость худшего случая (когда общая размерность нечётная).

Трудоёмкость алгоритма:

Худший случай: **12.5*n*N*m +7.5*m*N+7.5*n*N +27*n*m -3n + 5m + 10**

Лудший случай: **12.5*n*N*m +7.5*m*N+7.5*n*N +12*n*m -3n + m + 8**

Улучшенный алгоритм Винограда

$f_a = 2 + m * (2 + 7 + 2 + (N/21) * (2 + 10)) +$
$+ 2 + n * (2 + 7 + 2 + (N/21) * (2 + 10)) +$
$2 + m * (2 + n * (2 + 9 + 4 + 2 + N/2 * (2 + 18) + 3)) +$
$10 * N * n * m + 6 * N * m + 6 * N * n + m - n + 20 * n * m + 6]$

Трудоёмкость алгоритма:

Худший случай: **10*N*n*m + 6*N*m + 6*N*n + m-n+ 20*n*m +6**

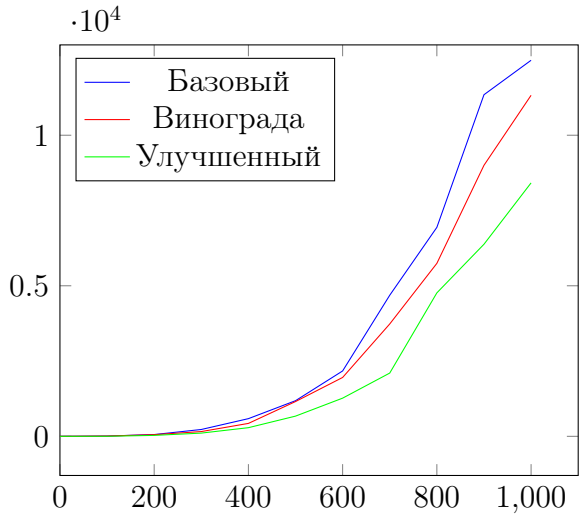
Лудший случай: **10N*n*m +6*N*m +6*N*n +m -n+8*n*m+6**

Эксперимент

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

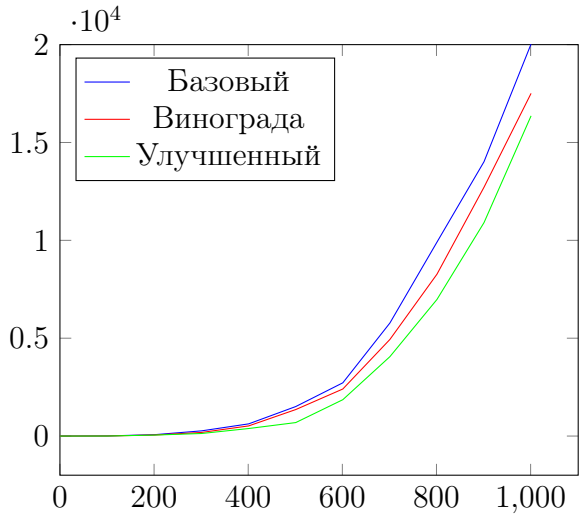
Матрицы чётной размерности

Размерность матрицы	**Базовый алгоритм**	**Алгоритм Винограда**	**Улучшенный алг. Винограда**
100x100	9	9	2
200x200	59	55	31
300x300	227	158	107
400x400	587	429	287
500x500	1183	1155	671
600x600	2171	1955	1268
700x700	4686	3732	2101
800x800	6940	5744	4764
900x900	11345	9001	6379
1000x1000	12489	11326	8417



Матрицы нечётной размерности

Размерность матрицы	Базовый алгоритм	Алгоритм Винограда	Улучшенный алг. Винограда
101x101	7	4	2
201x201	71	54	50
301x301	266	185	131
401x401	623	519	385
501x501	1508	1354	689
601x601	2720	2404	1858
701x701	5775	4935	4056
801x801	9907	8269	6978
901x901	14019	12715	10906
1001x1001	20034	17510	16362



Выводы из экспериментов

Улучшенный алгоритм Винограда быстрее стандартного примерно в 1.52 раза для матриц чётных размерностей, и в 1.3 раза для матриц нечётных размерностей.

Базовый алгоритм умножения матриц медленнее улучшенного алгоритма Винограда примерно в 1.8 раза для матриц чётных размерностей, и в 1.56 раза для матриц нечётных размерностей.

Заключение

В ходе лабораторной работы были изучены алгоритмы умножения матриц: стандартный и алгоритм Винограда, улучшен алгоритм Винограда, дана теоретическая оценка алгоритмов умножения матриц, а также произведено их сравнение.