Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **«ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ» (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

О Т Ч Е Т

по лабораторной работе № 2

Название: Трудоёмкость алгоритмов умножения матриц

Дисциплина: Анализ алгоритмов

Студент ИУ7-52Б _____ Е.В. Брянская
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Л.Л. Волкова
(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Оглавление

Введение

Трудоёмкость алгоритма - это зависимость стоимости операций от линейного(ых) размера(ов) входа(ов).

Модель вычислений трудоёмкости должна учитывать следующие оценки.

1) Стоимость базовых операций. К ним относятся: =, +, -, *, /, ==, !=, <, <=, >, >=, %, +=, -=, *=, /=, [ ], < <, > >. Каждая из операций имеет стоимость равную 1.

2) Оценка цикла. Она складывается из стоимости тела, инкремента и сравнения.

3) Оценка условного оператора if. Положим, что стоимость перехода к одной из веток равной 0. В таком случае, общая стоимость складывается из подсчета условия и рассмотрения худшего и лучшего случаев.

Оценка характера трудоёмкости даётся по наиболее быстрорастущему слагаемому.

В этой лабораторной работе будет оцениваться трудоёмкость алгоритмов умножения матриц.

1. Аналитическая часть

Цель данной работы – оценить трудоёмкость алгоритмов умножения матриц и получить практический навык оптимизации алгоритмов.

Для достижения поставленной цели необходимо решить ряд следующих **задач**:

1) дать математическое описание;

2) описать алгоритмы умножения матриц;

3) дать теоретическую оценку трудоёмкости алгоритмов;

4) реализовать эти алгоритмы ;

5) провести замеры процессорного времени работы алгоритмов на материале серии экспериментов;

6) провести сравнительный анализ алгоритмов.

Умножение осуществляется над матрицами $A[M \times N]$ и $B[N \times Q]$. Число столбцов первой матрицы должно совпадать с числом строк второй, а таком случае можно осуществлять умножение. Результатом является матрица $C[M \times Q]$, в которой число строк столько же, сколько в первой, а столбцов, столько же, сколько во второй.

В основе **стандартного алгоритма** умножения матриц лежит следующая формула:

$$c_{i,j} = \sum_{k=1}^{N}(a_{i,k} \times b_{k,j}) \tag{1.1}$$

Существует и другой алгоритм умножения - **алгоритм Винограда**. Обозначим строку $A_{i,*}$ как $\overrightarrow{u}$, $B_{*,j}$ как $\overrightarrow{v}$.

Пусть $u = (u_1, u_2, u_3, u_4)$ и $v = (v_1, v_2, v_3, v_4)$, тогда их произведение равно

$$u \cdot v = u_1 \cdot v_1 + u_2 \cdot v_2 + u_3 \cdot v_3 + u_4 \cdot v_4 \tag{1.2}$$

Выражение (1.2) можно преобразовать в следующее:

$$u \cdot v = (u_1 + v_2) \cdot (u_2 + v_1) + (u_3 + v_1) \cdot (u_4 + v_3) - u_1 \cdot u_2 - u_3 \cdot u_4 - v_1 \cdot v_2 - v_3 \cdot v_4 \tag{1.3}$$

Алгоритм Винограда основывается на раздельной работе со слагаемыми из выражения (1.3).

2. Конструкторская часть

Рассмотрим и оценим работу алгоритмов на матрицах $A[M \times N]$ и $B[N \times Q]$.

2.1. Стандартный алгоритм умножения матриц

В основе этого алгоритма лежит формула (1.1). То есть для вычисления произведения двух матриц, каждая строка первой матрицы почленно умножается на каждый столбец второй, и затем подсчитывается сумма таких произведений, и полученный результат записывается в соответствующую ячейку результурующей матрицы.

2.2. Алгоритм Винограда

Цель данного алгоритма - сократить долю умножений в самом тяжёлом, затратном участке кода. Для этого используется формула (1.3).

Некоторые из слагаемых можно вычислить заранее и использовать повторно для каждой строки первой матрицы и для каждого столбца второй. Таким образом, трудоёмкость алгоритма уменьшается за счёт сокращения количества производимых операций.

В этом алгоритме важно учитывать, что при нечётном значении N, необходимо вычислять дополнительное слагаемое $u_N \cdot v_N$.

2.3. Требования к ПО

Для корректной работы алгоритмов и проведения тестов необходимо выполнить следующее.

1) Обеспечить возможность ввода двух матриц через консоль и выбора алгоритма для умножения.

2) В случае ввода размеров матриц, не удовлетворяющих главному условию, вывести соответствующее сообщение. Программа не должна аварийно завершаться.

3) Программа должна рассчитать искомую матрицу и вывести её на экран.

4) Реализовать функцию замера процессорного времени, которое выбранный метод затрачивает на вычисление результата. Дать возможность пользователю ввести размер рассматриваемых матриц через консоль. Вывести результаты замеров на экран.

2.4. Заготовки тестов

При проверке на корректность работы реализованных функций необходимо провести следующие тесты:

- умножение матриц размером 1×1;
- квадратные матрицы;
- прямоугольные матрицы;
- чётное и нечётное значение N.

3. Технологическая часть

3.1. Выбранный язык программирования

Для выполнения этой лабораторной работы был выбран язык программирования C++, так как есть большой навык работы с ним и с подключаемыми библиотеками, которые также использовались для проведения тестирования и замеров.

Использованная среда разработки - Visual Studio.

3.2. Листинг кода

Ниже представлены Листиги 3.1 - 3.3 функций, реализующих алгоритмы поиска расстояний.

Листинг 3.1: Стандартный алгоритм умножения матриц

```
matrix_t standart_mult(matrix_t a, matrix_t b, int m, int n, int q)
{
  matrix_t c = create_matrix(m, q);

  for (int i = 0; i < m; i++)
    for (int j = 0; j < q; j++)
    {
      c[i][j] = 0;
      for (int k = 0; k < n; k++)
        c[i][j] += a[i][k] * b[k][j];
    }

  return c;
}
```

Листинг 3.2: Алгоритм Винограда

```
matrix_t winograd_mult(matrix_t a, matrix_t b, int m, int n, int q)
{
  arr_t mulH = create_array(m);
  arr_t mulV = create_array(n);
  matrix_t c = create_matrix(m, q);

  for (int i = 0; i < m; i++)
  {
```

```
    mulH[i] = 0;
    for (int k = 0; k < n / 2; k++)
      mulH[i] = mulH[i] + a[i][2 * k] * a[i][2 * k + 1];
  }

  for (int i = 0; i < q; i++)
  {
    mulV[i] = 0;
    for (int k = 0; k < n / 2; k++)

      mulV[i] = mulV[i] + b[2 * k][i] * b[2 * k + 1][i];
  }

  for (int i = 0; i < m; i++)
    for (int j = 0; j < q; j++)
    {
      c[i][j] = -mulH[i] - mulV[j];
      for (int k = 0; k < n / 2; k++)
        c[i][j] = c[i][j] + (a[i][2 * k] + b[2 * k + 1][j]) *
                  (a[i][2 * k + 1] + b[2 * k][j]);
    }

  if (n % 2)
    for (int i = 0; i < m; i++)
      for (int j = 0; j < q; j++)
        c[i][j] = c[i][j] + a[i][n - 1] * b[n - 1][j];

  return c;
}
```

Листинг 3.3: Оптимизированный алгоритм Винограда

```
matrix_t winograd_mult(matrix_t a, matrix_t b, int m, int n, int q)
{
  arr_t mulH = create_array(m);
  arr_t mulV = create_array(n);
  double buf;

  matrix_t c = create_matrix(m, q);

  for (int i = 0; i < m; i++)
```

```
10    {
11      buf = 0;
12      for (int k = 1; k < n; k += 2)
13        buf += a[i][k] * a[i][k - 1];
14      mulH[i] = buf;
15    }
16
17    for (int i = 0; i < q; i++)
18    {
19      buf = 0;
20      for (int k = 1; k < n; k += 2)
21        buf += b[k][i] * b[k - 1][i];
22      mulV[i] = buf;
23    }
24
25    int temp = n - 1;
26
27    for (int i = 0; i < m; i++)
28      for (int j = 0; j < q; j++)
29      {
30        buf = -(mulH[i] + mulV[j]);
31        for (int k = 1, t = 0; k < n; k += 2, t +=2)
32          buf += (a[i][k] + b[t][j]) * (a[i][t] + b[k][j]);
33        c[i][j] = buf;
34
35        if (n % 2)
36          c[i][j] += a[i][temp] * b[temp][j];
37      }
38
39    return c;
40  }
```

3.3. Оптимизации алгоритма Винограда

Чтобы уменьшить трудоёмкость алгоритма были использованы следующие оптимизации.

1) Видоизменён цикл по k, изменён шаг и условие. Таким образом, ушла необходимость в целочисленном делении, и в теле цикла не требуется больше умножать k на 2 каждый раз.

2) Введена вспомогательная переменная buf, в которую записывается промежуточнее значение соответсвующей ячейки матрицы, и затем, конечный результат переносится в саму матрицу. Тем самым, уменьшается количество обращений к элементам матрицы, находящимся по конкретному адресу.

3) Заранее высчитываются некоторые значения, например, n - 1, которые далее используюся во вложенных циклах.

4) Используется дополнительная переменная t = k - 1, чтобы сократить число подсчетов этого значения на каждом шаге цикла.

5) Объединён цикл 3 и 4, что позволило избежать ещё одного сложенного цикла.

3.4. Результаты тестов

3.5. Оценка трудоёмкости

Исследовательская часть

Заключение