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»

Лабораторная работа № 2
по дисциплине «Анализ Алгоритмов»

Тема Алгоритмы умножения матриц

Студент Шахнович Дмитрий Сергеевич

Группа ИУ7-52Б

Преподаватели Волкова Л.Л., Строганов Д.М.

Москва, 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Умножение матриц [**?**] – частая задача в программировании, так как матрицы используются для представления многих других математических объектов, например:

— системы уравнений;

— графы;

— преобразования пространственных координат;

— и другие.

При этом для работы с ними как одна из операций используется матричное умножение. Поэтому важно делать эту частую операцию эффективной.

Целью данной лабораторной работы является изучение стандартного алгоритма умножения матриц и алгоритма Винограда.

Задачами данной лабораторной являются:

1) рассмотрение алгоритмов стандартного умножения матриц и алгоритма Винограда;
2) разработка стандартного алгоритма умножения матриц и алгоритма Винограда;
3) оптимизация алгоритма Винограда с помощью замены некоторых операций;
4) оценка трудоёмкости разработанных алгоритмов;
5) реализация разработанных алгоритмов;
6) исследование временных характеристик реализованных алгоритмов.

1 Аналитическая часть

1.1 Теоретические сведения по матрицам

Матрицей [1, 390 с.] размера m на n называется таблица некоторых элементов a_{jk}, которая задаётся в виде:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \tag{1.1}$$

При этом такую матрицу можно сокращённо записать:

$$A = [a_{jk}], \quad 0 \leq j \leq m, 0 \leq k \leq n. \tag{1.2}$$

Элементы a_{jk} называют элементами матрицы.

Если для элементов матрицы определена операция умножения, то можно ввести умножение матриц [1, 392 с.] следующим образом: Произведение матрицы $A = [a_{ik}]$ размера m на n на матрицу $B = [b_{kj}]$ размера n на r есть матрица $C = [C_{ij}]$ размера m на r, при этом элементы этой матрицы определяются по формуле 1.3:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} * b_{kj}, \quad 0 \leq i \leq m, 0 \leq j \leq r. \tag{1.3}$$

1.2 Стандартный метод умножения матриц

Стандартный алгоритм построен на определении матричного умножения, без каких-либо усложнений или ухищрений. В данном методе результирующая матрица рассчитывается поэлементно по формуле 1.3.

1.3 Алгоритм Винограда

Можно заметить, что в формуле 1.3 элемент матрицы рассчитывается как скалярное произведение соответствующих строки на столбец исходных матриц. Скалярное произведение определяется как:

$$\vec{V}\vec{W} = V_1W_1 + V_2W_2 + \cdots + V_nW_n, \tag{1.4}$$

где n – размер векторов.

Скалярное произведение 1.4 можно записать в виде 1.5 для чётных n и 1.6 – для нечётных n

$$\vec{V}\vec{W} = (V_1 + W_2)(V_2 + W_1) + \cdots + (V_{n-1} + W_n)(V_n + W_{n-1}) - \\ -V_1V_2 - W_1W_2 - \cdots - V_{n-1}V_n - W_{n-1}W_n \tag{1.5}$$

$$\vec{V}\vec{W} = (V_1 + W_2)(V_2 + W_1) + \cdots + (V_{n-2} + W_{n-1})(V_{n-1} + W_{n-2}) - \\ -V_1V_2 - W_1W_2 - \cdots - V_{n-2}V_{n-1} - W_{n-2}W_{n-1} + V_nW_n \tag{1.6}$$

В этих формулах интересно то, что их части можно посчитать заранее для каждой из матриц, и для первой хранить значения для каждой строки, а для второй – для каждого столбца. За счёт этого алгоритм может получится эффективнее, ценой дополнительной памяти на хранение значений для матриц.

Вывод

В результате аналитического раздела были рассмотрены математические основы стандартного алгоритма умножения матриц и алгоритма Винограда.

2 Конструкторская часть

2.1 Требования к программному обеспечению

К разрабатываемой программе предъявлен ряд требований.

Входные данные: Две матрицы, подходящие для умножения размеров

Выходные данные: Матрица, являющаяся их произведением

2.2 Разработка алгоритмов

Пусть в качестве входных данных алгоритмам подаются матрицы A и B размерами M на N и N на Q элементов соответственно. На листинге 2.1 показан псевдокод стандартного алгоритма умножения матриц, на листинге 2.2 – алгоритм винограда.

Листинг 2.1 — Псевдокод стандартного алгоритма умножения

```
for (i=0; i < M; i++) {
  for (j=0; j < Q; j++) {
    C[i][j] = 0
    for (k=0; k < N; k++) {
      C[i][j] = C[i][j] +
      A[i][k]*B[k][j];
    }
  }
}
```

Листинг 2.2 — Псевдокод алгоритма Винограда

```
  // I. Заполнение массива mulH
  for (i=0; i < M; i++) {
    for (k = 0; k < N/2; k++) {
      mulH[i] = mulH[i] +
      A[i][2*k] * A[i][2*k + 1]
    }
  }

  // II. Заполнение mulV
  for (i=0; i < Q; i++) {
    for (k = 0; k < N/2; k++) {
      mulV[i] = mulV[i] +
      B[2*k][i] * B[2*k + 1][i]
    }
  }
```

```
// III Основная часть
for (i=0; i < m; i++) {
  for (j=0; j < Q; j++) {
    C[i][j] = -Mulh[i] - MulV[j]
    for (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]);
    }
  }
}

// IV Обработка на случай нечётной общей размерности матриц
if (N % 2 != 0) {
  for (i=0; i < m; i++) {
    for (j=0; j < Q; j++) {
      C[i][j] = C[i][j] + A[i][N-1] * B[N-1][j]
    }
  }
}
```

В виде, представленном на листинге 2.2, используется много неэффективных решений, в частности возможны следующие оптимизации:

— во вложенных циклах замена шага на 2, а условие цикла до N;

— замена присваивания со сложением в телах циклов на инкремент;

— в основной части, в самом вложенном цикле вынесение первой итерацию во вне цикла.

На листинге 2.3 можно увидеть псевдокод алгоритма Винограда с описанными выше оптимизациями.

Листинг 2.3 — Псевдокод алгоритма Винограда с оптимизациями

```
// I. Заполнение массива mulH
for (i = 0; i < M; i++) {
  for (k = 0; k < N; k += 2) {
    mulH[i] += A[i][k] * A[i][k + 1]
  }
}

// II. Заполнение mulV
for (i = 0; i < Q; i++) {
  for (k = 0; k < N; k += 2) {
    mulV[i] += B[k][i] * B[k + 1][i]
  }
```

```
  }


  // III Основная часть
  for (i = 0; i < m; i++) {
    for (j = 0; j < Q j++) {
      C[i][j] = -Mulh[i] - MulV[j] + (A[i][0]+B[1][j]) * (A[i][1]+B[0][j])
      for (k = 2; k < N; k += 2) {
        C[i][j] += (A[i][k]+B[k + 1][j]) * (A[i][k + 1]+B[k][j]);
      }
    }
  }

  // IV Обработка на случай нечётной общей размерности матриц
  if (N % 2 != 0) {
    for (i=0; i < m; i++) {
      for (j=0; j < Q j++) {
        C[i][j] += A[i][N-1] * B[N-1][j]
      }
    }
  }
```

2.3 Модель вычислений

Для оценки трудоёмкости разработанных алгоритмов введём модель вычислений.

2.3.1 Трудоёмкость базовых операций

Примем единичной трудоёмкость следующих операций: =, +, -, +=, -=, ==, !=, <, <=, >=, >, [], «, ». Трудоёмкость 2 имеют следующие операции: * /, *=, /=, %.

2.3.2 Трудоёмкость цикла

Пусть для цикла вида:

```
for (инициализация; сравнение; инкремент) {
  тело цикла
},
```
(2.1)

известны трудоёмкости блоков инициализации, сравнения, инкремента и тела и они соответственно равны $f_{init}, f_{comp}, f_{inc}, f_{body}$. Тогда трудоёмкость цикла рассчитывается по формуле 2.2:

$$f_{cycle} = f_{init} + f_{comp} + M(f_{comp} + f_{inc} + f_{body}), \tag{2.2}$$

где M - количество итераций выполненных циклом.

2.3.3 Трудоёмкость условного оператора

Пусть для условного оператора вида:

```
if (условие) {
   тело1
} else {
   тело2
},
```
(2.3)

известны трудоёмкости блоков условия, тел 1 и 2 и они соответственно равны f_{cond}, f_1, f_2. Тогда трудоёмкость условного оператора рассчитывается по формуле 2.4:

$$f_{if} = f_{cond} + \begin{cases} min(f1, f2), & \text{в лучшем случае,} \\ max(f1, f2), & \text{в худшем случае.} \end{cases} \tag{2.4}$$

2.4 Расчёт трудоёмкости алгоритмов

Трудоёмкости для разработанных алгоритмов будем рассматривать в лучшем и худшем случаях относительно их размеров M, N и Q.

Для стандартного алгоритма худший случай совпадает с лучшим, так как в нём нет условных операторов. Его трудоёмкость считается следующим образом:

$$f_{std} = 14MNQ + 7MQ + 4M + 2 \tag{2.5}$$

Слагаемые приведены в формуле 2.5 в порядке следования в программе 2.1.

Для алгоритма винограда худшим случаем будет, когда общая размерность матриц будет нечётной, так как тогда будет дополнительно выполняться тело 4-й части программы 2.2. Расчёт трудоёмкости всех частей алгоритма винограда приведён на формулах 2.6–2.9.

$$f_I = \tfrac{19}{2}MN + 6M + 2 \tag{2.6}$$

2-я часть программы 2.2 от 1-ой с точки зрения трудоёмкости отличается только переменной внешнего цикла, поэтому:

$$f_{II} = \frac{19}{2}QN + 6Q + 2 \tag{2.7}$$

$$f_{III} = 16MNQ + 13MQ + 4M + 2 \tag{2.8}$$

$$f_{IV} = \begin{cases} 3, & \text{если N – чётно,} \\ 16MQ + 4M + 5, & \text{если N – нечётно.} \end{cases} \tag{2.9}$$

Тогда общая трудоёмкость алгоритма Винограда будет иметь вид:

$$f_V = \begin{cases} 16MNQ + 13MQ + \frac{19}{2}N(M+Q) + 10M + 6Q + 9, & \text{если N – чётно,} \\ 16MNQ + 29MQ + \frac{19}{2}N(M+Q) + 14M + 6Q + 11, & \text{если N – нечётно.} \end{cases} \tag{2.10}$$

Порядок трудоёмкости, то есть сложность алгоритма, определяется через самое быстрорастущее слагаемое, то есть и для стандартного и для алгоритма Винограда это MNQ, при это коэффициент при этом слагаемом в алгоритме винограда выше, а значит он менее эффективен.

Рассмотрим оптимизированный алгоритм по частям, аналогично обычному алгоритму Винограда. Трудоёмкости приведены на 2.11–2.14

$$f_I^o = \frac{11}{2}MN + 4M + 2 \tag{2.11}$$

$$f_{II}^o = \frac{11}{2}QN + 4Q + 2 \tag{2.12}$$

$$f_{III}^o = \frac{19}{2}MNQ + 5MQ + 4M + 2 \tag{2.13}$$

$$f_{IV}^o = \begin{cases} 3, & \text{если N – чётно,} \\ 13MQ + 4M + 5, & \text{если N – нечётно.} \end{cases} \tag{2.14}$$

Итоговая трудоёмкость оптимизированного алгоритма:

$$f_V^o = \begin{cases} \frac{19}{2}MNQ + 5MQ + \frac{11}{2}N(M+Q) + 8M + 4Q + 9, & \text{если N – чётно,} \\ \frac{19}{2}MNQ + 18MQ + \frac{11}{2}N(M+Q) + 12M + 4Q + 11, & \text{если N – нечётно.} \end{cases} \tag{2.15}$$

Таким образом оптимизированный алгоритм винограда должен быть эффективнее стандартного

2.5 Вывод

В результате конструкторской части были определены требования к ПО, а также разработаны псевдокоды алгоритмов стандартного умножения, умножения Винограда, проведена оптимизация умножения Винограда и рассчитаны трудоёмкости алгоритмов.

3 Технологическая часть

3.1 Средства разработки

В качестве языка программирования был выбран python3 [2], так как в его стандартной библиотеке присутствуют функции замера процессорного времени, которые требуются в условиях, а также данный язык обладает множеством инструментов для работы с данными. В частности была взята его реализация micropython [3], которая разработана для работы с микроконтроллерами, на которых планировалось проводить замеры.

Для файла с графиком был выбран инструмент jupyter notebook [4], так как он позволяет организовать код в удобные блоки, а также выводить данные и графики прямо в нём, что позволяет легко продемонстрировать все замеры.

Для построения графиков использовалась библиотека plotly [6].

Для замера времени использовалась функция ticks_ms() из стандартного модуля utime [5] для micropython.

3.2 Реализация алгоритмов

В листингах 3.1–3.2 приведены реализации разработанных в конструкторской части алгоритмов(рисунки 2.1–3.3).

Листинг 3.1 — Стандартный алгоритм умножения матриц

```
def SimpleMatrixMultiply(m1: list[list[int]], m2: list[list[int]]):
  if len(m1[0]) != len(m2):
    raise ValueError("Matrices cannot be multiplied")

  result = [[0] * len(m2[0]) for _ in range(len(m1))]

  for i in range(len(m1)):
    for j in range(len(m2[0])):
      for k in range(len(m2)):
        result[i][j] += m1[i][k] * m2[k][j]

  return result
```

Листинг 3.2 — Алгоритм умножения матриц Винограда

```
def VinogradMatrixMultiply(m1: list[list[int]], m2: list[list[int]]):
  if (len(m1) == 0 or len(m2) == 0):
    raise ValueError("Empty matrix")

  if len(m1[0]) != len(m2):
    raise ValueError("Matrices cannot be multiplied")
```

```
    M = len(m1)
    N = len(m1[0]) # == len(m2)
    Q = len(m2[0])
    result = [[0] * Q for _ in range(M)]

    mulH = [0] * (M)
    for i in range(M):
      for j in range(N // 2):
        mulH[i] = mulH[i] + m1[i][2*j] * m1[i][2*j + 1]

    mulV = [0] * (Q)
    for i in range(Q):
      for j in range(N // 2):
        mulV[i] = mulV[i] + m2[2*j][i] * m2[2*j + 1][i]

    for i in range(M):
      for j in range(Q):
        result[i][j] = -mulH[i] -mulV[j]
        for k in range(N // 2):
          result[i][j] = result[i][j] + (m1[i][2*k]+m2[2*k + 1][j]) * (m1[
              i][2*k + 1]+m2[2*k][j])

    if (N % 2 != 0):
      for i in range(M):
        for j in range(Q):
          result[i][j] = result[i][j] + m1[i][-1] * m2[-1][j]

    return result
```

Листинг 3.3 — Оптимизированный алгоритм умножения матриц Винограда

```
  def OptimizedVinogradMatrixMultiply(m1: list[list[int]], m2: list[list[
      int]]):
    if (len(m1) == 0 or len(m2) == 0):
      raise ValueError("Empty matrix")

    if len(m1[0]) != len(m2):
      raise ValueError("Matrices cannot be multiplied")

    M = len(m1)
    N = len(m1[0]) # == len(m2)
    Q = len(m2[0])
```

```
    result = [[0] * Q for _ in range(M)]

    mulH = [0] * (M)
    for i in range(M):
      mulH[i] = m1[i][0] * m1[i][1]
      for j in range(2, N - 1, 2):
        mulH[i] += m1[i][j] * m1[i][j + 1]

    mulV = [0] * (Q)
    for i in range(Q):
      mulV[i] = m2[0][i] * m2[1][i]
      for j in range(2, N - 1, 2):
        mulV[i] += m2[j][i] * m2[j + 1][i]

    for i in range(M):
      for j in range(Q):
        result[i][j] = -mulH[i] -mulV[j] + (m1[i][0]+m2[1][j]) * (m1[i
           ][1]+m2[0][j])
        for k in range(2, N - 1, 2):
          result[i][j] += (m1[i][k]+m2[k + 1][j]) * (m1[i][k + 1]+m2[k][j
             ])

    if (N % 2 != 0):
      for i in range(M):
        for j in range(Q):
          result[i][j]  += m1[i][-1] * m2[-1][j]

    return result
```

Вывод

В ходе технологической части работы были реализованы алгоритмы умножения матриц разработанные в конструкторской части.

4 Исследовательская часть

4.1 Технические характеристики

Замеры проводились на микроконтроллере STM32F767 Nucleo-144 с установленным micropython.

Микроконтроллер имеет процессор Arm Cortex-M7 216 МГц и 512 КБ оперативной памяти.

4.2 Временные характеристики

Временные характеристики алгоритмов замерялись при вариативном линейном размере квадратных матриц, при этом каждый размер матрицы считался количество раз, которое позволяла память платы. Количество измерений для каждого размера указано в последнем столбце таблицы. На каждое измерение генерировались две случайные матрицы заданного размера и подавались на вход всем 3-м функциям. Результаты замеров представлены в таблице 4.1

Таблица 4.1 — Временные характеристики

Размер, элементы	Стандартный алгоритм, мс	Алгоритм Винограда, мс	Оптимизированный алгоритм Винограда, мс	Количество замеров
5	1	1	1	20
10	8	8	6	20
15	26	26	21	20
20	61	61	48	20
25	117	120	96	20
30	201	212	170	10
35	318	338	274	10
40	474	514	421	5
45	671	752	619	5
50	919	1010	828	3

Полученные замеры также можно увидеть на графике (4.1):

Как видно из таблицы (4.1) и графика (4.1), оптимизированный алгоритм эффективнее двух других.

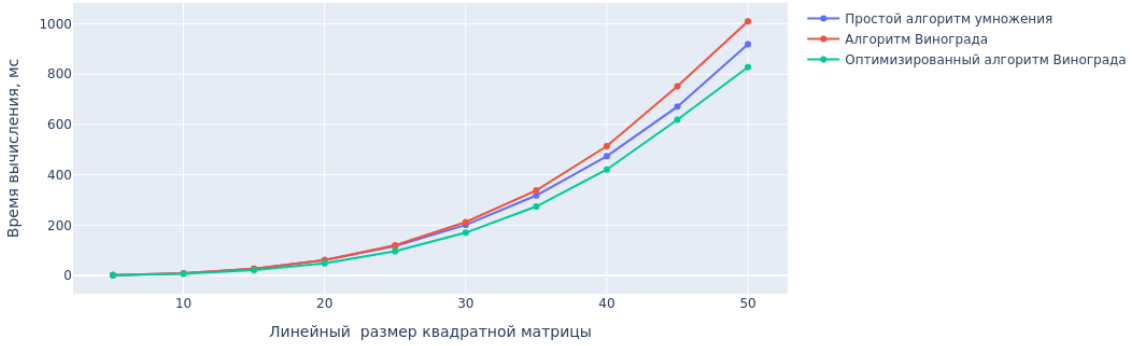

Рисунок 4.1 — График зависимости времени работы алгоритмов от линейного размер квадратной матрицы

4.3 Вывод

В данном разделе было проведено исследование временных и алгоритмов умножения матриц.

В результате исследования было выявлено, что самым эффективным оказался алгоритм Винограда с оптимизацией, при этом алгоритм Винограда без оптимизации оказался самым медленным. Результаты исследования соотносятся с трудоёмкостями алгоритмов, рассчитанным в формулах 2.5–2.15.

ЗАКЛЮЧЕНИЕ

Цель – изучить алгоритмы умножения матриц – выполнена. При этом в ходе работы были выполнены следующие задачи:

— рассмотрены алгоритмы стандартного умножения матриц и Винограда;

— разработаны стандартный алгоритм умножения матриц и алгоритм умножения матриц Винограда;

— оптимизирован алгоритм Винограда заменой некоторых операций и конструкций;

— проведена оценка трудоёмкости разработанных алгоритмов;

— реализованы разработанные алгоритмы на языке python3;

— проведено исследование временных характеристик реализованных алгоритмов.

В результате исследования было выявлено, что классический алгоритм Винограда является менее эффективным чем стандартный алгоритм, однако при небольших оптимизациях, он начинает обгонять стандартный алгоритм. В итоге самым эффективным во времени оказался оптимизированный алгоритма винограда, а самым не эффективным – классический алгоритм Винограда. При этом эти результаты соотносятся с проведённой оценкой трудоёмкости.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Корн Г., Корн Т. Справочник по математике [Текст] / Корн Г., Корн Т. — Москва: Наука, 1974 — 832 с.

2. Python / [Электронный ресурс] // Python : [сайт]. — URL: https://www.python.org/ (дата обращения: 20.09.2024).

3. MicroPython / [Электронный ресурс] // MicroPython : [сайт]. — URL: https://micropython.org/ (дата обращения: 08.10.2024).

4. Jupyter Notebook: The Classic Notebook Interface / [Электронный ресурс] // Jupyter : [сайт]. — URL: https://jupyter.org/ (дата обращения: 20.09.2024).

5. utime – time related functions / [Электронный ресурс] // MicroPython documentation : [сайт]. — URL: https://docs.micropython.org/en/v1.15/library/utime.html (дата обращения: 08.10.2024).

6. Plotly Open Source Graphing Library for Python / [Электронный ресурс] // Plotly : [сайт]. — URL: https://plotly.com/python/ (дата обращения: 20.09.2024).