Государственное образовательное учреждение высшего профессионального образования «Московский Государственный Технический Университет имени Н.Э. Баумана»

Отчет

По лабораторной работе № 2
По курсу «Анализ Алгоритмов»
На тему «Исследование сложности алгоритмов умножения матриц»

Студент: Козырев Марк
Группа: ИУ7-56
Преподаватель: Волкова Л. Л.

Москва, 2018

Введение

В лабораторной работе изучаются алгоритмы умножения матриц. Рассмотрены алгоритмы: стандартный, алгоритм Винограда и улучшенный алгоритм Винограда.

Цель лабораторной работы - проанализировать трудоемкость алгоритма Винограда и исследовать возможности его улучшения и провести сравнительный анализ времени работы алгоритма для различных размеров матриц

Постановка задачи

1. Ввести модель оценки трудоемкости
2. Реализовать алгоритмы умножения матриц:
    1. классический алгоритм умножения
    2. алгоритм Винограда
    3. улучшенный алгоритм Винограда (провести не менее 3-х улучшений)
3. Провести временные замеры
4. Провести рассвет трудоемкости для реализованных алгоритмов
5. Провести сравнительный анализ времени работы алгоритма для разных исходных матриц

1. Аналитическая часть

В данном разделе приведены алгоритмы и составлена модель для вычисления трудоемкости.

1. 1. Описание алгоритмов

В данном разделе приведены описание алгоритмов и их работы

1. 1. 1. Стандартный алгоритм умножения

Имеем две матрицы A и B размерностями M x N и N x Q соответственно. Тогда результирующей матрицей будет матрица C с размером M x Q, где $c_{ij} = \sum_{r=1}^{n} a_{ir} \cdot b_{rj}, (i = 0,1,2...m, j = 0,1,2...q)$.

1. 1. 2. Алгоритм Винограда

Алгоритм Винограда разделяется на 4 части:

Пусть i-ая строка матрицы A - вектор $\overrightarrow{U}$, а j-ый столбец матрицы B - вектор $\overrightarrow{V}$. Тогда $C_{ij} = (u_1 + v_2) \cdot (u_2 \cdot v_1) + (u_3 + v_4) \cdot (u_4 + v_3) - u_1u_2 - u_3u_4 - v_1v_2 - v_3v_4$.

- Предварительный расчет сумм произведений пар элементов каждой строки матрицы A: $u_1u_2 - u_3u_4 - v_1v_2 - v_3v_4$
- Аналогичные вычисления для столбцов B
- Вычисление C_{ij}
- Если вектора u и v нечетной длины, тогда добавляем: $C_{ij} \mathrel{+}= U_{N-1} \cdot V_{N-1}, \forall i, j$

1. 2. Модель вычислений

Введем следующую модель вычислений:
операции + - ; / < <= > >= = <> [] % имеют стоимость 1.

1. 2. 1. Оценка трудоемкости цикла for

Инициализация до цикла стоит 2, после выполнения тела цикла, инкрементируется итератор цикла, проверяется условие по стоимости выше. Если в условии содержится выражение, то оно считается как сумма стоимости простых операций выше.

1. 2. 2. Оценка трудоемкости операции if

Переход по условию имеет стоимость 0, проверка условия зависит от выражения самой провеки согласно модели выше.

2. Конструкторская часть

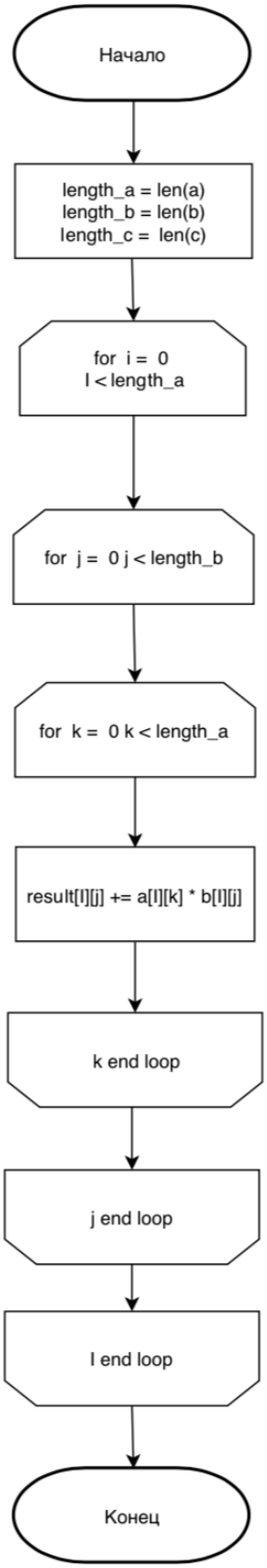

Рис. 1: Схема стандартного алгоритма

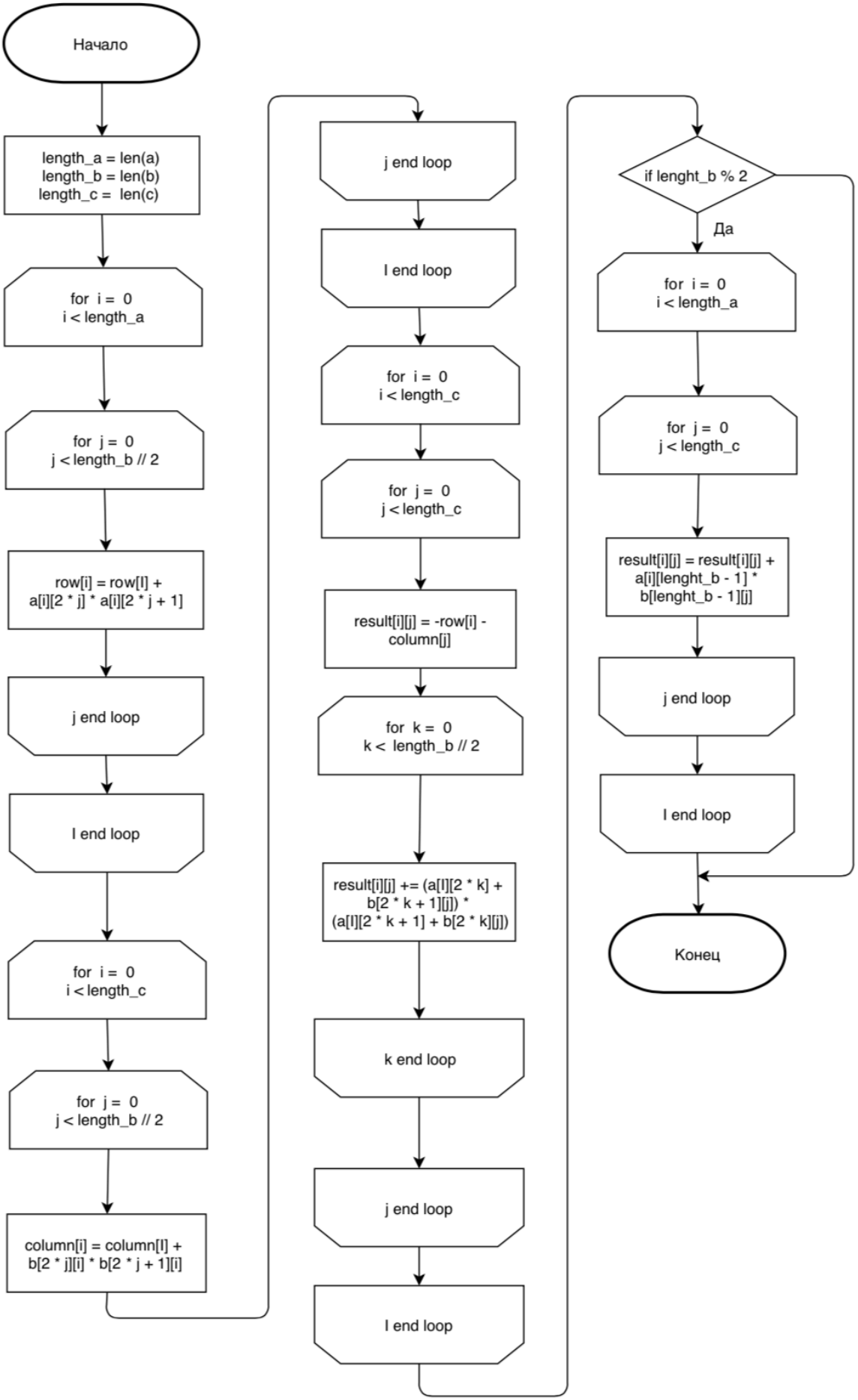

Рис. 2: Схема алгоритма Винограда

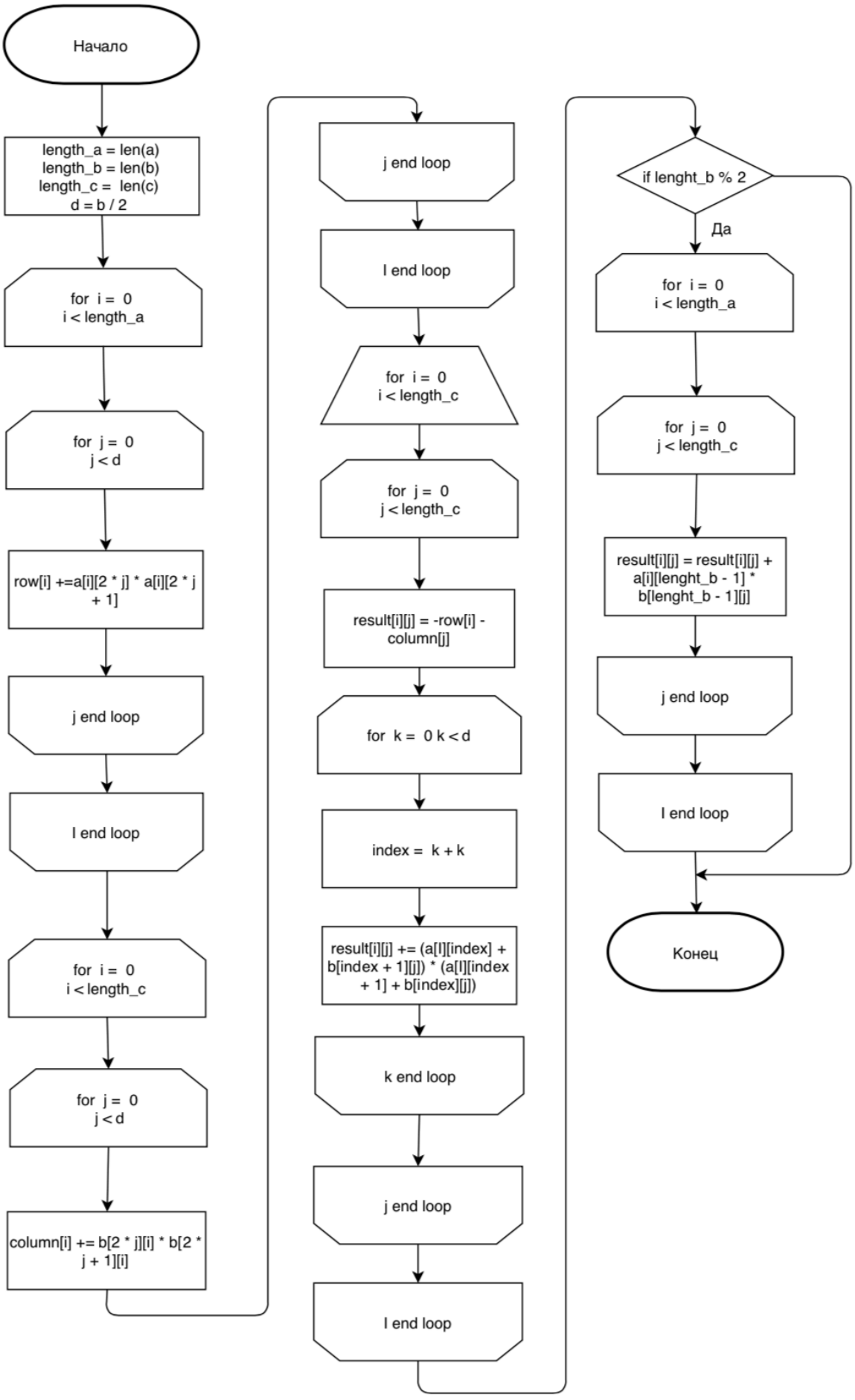

Рис. 3: Схема улучшенного алгоритма Винограда

3. Технологическая часть

В этом разделе приведена реализация функций, указан язык программирования и необходимые модули.

3. 1. Средства реализации

В данной работе использовался язык Python 3.6. Для измерения времени использовался модуль time, измерения производились в миллисекундах.

3. 2. Листинг кода

```
import time

def main():
  a = [[i for i in range(1000)] for j in range(1000)]
  b = [[i for i in range(1000)] for j in range(1000)]
  start1 = time.time()
  print(multiplication(a, b))
  end1 = time.time()
  start2 = time.time()
  print(vineyard(a, b))
  end2 = time.time()
  start3 = time.time()
  print(grapefull_vineyard(a, b))
  end3 = time.time()

  print(end1 - time1)
  print(end2 - time2)
  print(end3 - time3)

def multiplication(a, b):
  length_a = len(a)
  length_b = len(b)
  result = [[0 for i in range(1000)] for j in range(1000)]
  for i in range(length_a):
    for j in range(length_b):
      for k in range(length_a):
        result[i][j] += a[i][k] * b[k][j]
  return result

def vineyard(a, b):
  lenght_a = len(a)
  lenght_b = len(b)
  lenght_c = len(b[0])

  result = [[0 for i in range(lenght_c)] for j in range(lenght_a)]
  row = [0 for i in range(lenght_a)]
  column = [0 for i in range(lenght_c)]

  for i in range(lenght_a):
    for j in range (0, lenght_b // 2):
      row[i] = row[i] + a[i][2 * j] * a[i][2 * j + 1]

  for i in range(lenght_c):
    for j in range(0, lenght_b // 2):
      column[i] = column[i] + b[2 * j][i] * b[2 * j + 1][i]

  for i in range(lenght_a):
    for j in range(lenght_c):
      result[i][j] = -row[i] - column[j]
      for k in range(lenght_b // 2):
        result[i][j] = result[i][j] + (a[i][2 * k] + b[2 * k + 1][j
    ]) * (a[i][2 * k + 1] + b[2 * k][j])

  if lenght_b % 2:
    for i in range(lenght_a):
      for j in range(0, lenght_c):
        result[i][j] = result[i][j] + a[i][lenght_b - 1] * b[
```

```
    lenght_b - 1][j]

  return result


def grapefull_vineyard(a, b):
  lenght_a = len(a)
  lenght_b = len(b)
  lenght_c = len(b[0])

  d = lenght_b // 2
  result = [[0 for i in range(lenght_c)] for j in range(lenght_a)]
  row = [0 for i in range(lenght_a)]
  column = [0 for i in range(lenght_c)]

  for i in range(lenght_a):
    for j in range (d):
      row[i] += a[i][2 * j] * a[i][2 * j + 1]

  for i in range(lenght_c):
    for j in range(d):
      column[i] += b[2 * j][i] * b[2 * j + 1][i]

  for i in range(lenght_a):
    for j in range(lenght_c):
      result[i][j] = -row[i] - column[j]
      for k in range(d):
        index = 2 * k
        result[i][j] += (a[i][index] + b[index + 1][j]) * (a[i][
    index + 1] + b[index][j])

  if lenght_b % 2:
    for i in range(lenght_a):
      for j in range(0, lenght_c):
        result[i][j] += a[i][lenght_b - 1] * b[lenght_b - 1][j]

  return result


if __name__ == '__main__':
  main()
```

main.py

3. 3. Вычисление трудоемкости алгоритмов

Расчет производился по исходному коду. Разделение на части проводилось согласно логическим сегментам программы. Для сокращения времени работы алгоритма были сделаны следующие улучшения:

1. До начала работы алгоритма введена новая переменная границы цикла d = length_b // 2.
2. Введен оператор + = для сокращения количества операций.
3. Введена новая переменная index = 2 * k, для сокращения кол-ва операций умножения в индексах.

Оценка трудоемкости стандартного алгоритма:

$10MNQ + 4MQ + 4M + 2$

Оценка трудоемкости алгоритма Винограда

Первая часть: $\frac{13MN}{2} + 5M + 2$

Вторая часть: $\frac{13QN}{2} + 5M + 2$

Третья часть: $13MNQ + 9MQ + 2M + 2$

Четвертая часть: $15QM + 2M + 1$

Оценка трудоемкости улучшенного алгоритма Винограда

Первая часть: $6MN + 2M + 2$

Вторая часть: $6QN + 2M + 2$

Третья часть: $10MNQ + 9QM + 2M + 2$

Четвертая часть: $12MQ + 2M + 1$

4. Экспериментальная часть

В данном разделе будут приведены примеры работы алгоритмов и произведены замеры времени. Замеры времени проводились на компьютере с процессоре 2,4 GHz Intel Core i5 и с оперативной памятью 8 ГБ 1600 MHz DDR3

4. 1. Примеры работы

Имеем матрицы A с размером a x b и матрицу B с размером a x b, результатом алгоритмов будет матрица C с размером a x b:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{bmatrix}$$

4. 2. Временные эксперименты

Измерения проводились для квадратных целочисленных матриц с помощью функцией time() из встроенного модуля python time. Замеры времени в миллисекундах.

Размер	Обычное	Виноград	Улучшенный Виноград
100 X 100	290.94333	361.24867	307.45867
200 X 200	2353.78200	2815.53167	2542.62967
300 X 300	7412.31867	8559.82133	7807.55367
400 X 400	18560.03933	22043.19033	19563.29733
500 X 500	37094.78200	44213.58000	41490.46667
600 X 600	63905.64467	82005.13800	72646.38367
700 X 700	107256.68167	137257.87933	123870.87800
800 X 800	154546.39333	196856.36467	176563.80400
900 X 900	221015.69867	280759.76167	258684.84033
1000 X 1000	306991.44867	386513.90867	359238.27967
101 X 101	303.88400	356.69100	312.80333
201 X 201	2230.60500	2600.42033	2351.83467
301 X 301	7662.25700	8809.80667	8061.61600
401 X 401	18658.00100	22335.59000	21140.32800
501 X 501	37049.78233	43786.63667	40248.24167
601 X 601	64789.63533	78651.74167	71082.76400
701 X 701	103867.78933	125626.53600	117352.42733
801 X 801	155306.45600	198218.44867	175573.72567
901 X 901	220950.60200	274289.26800	254960.42733
1001 X 1001	305296.35733	390898.75233	345251.03733

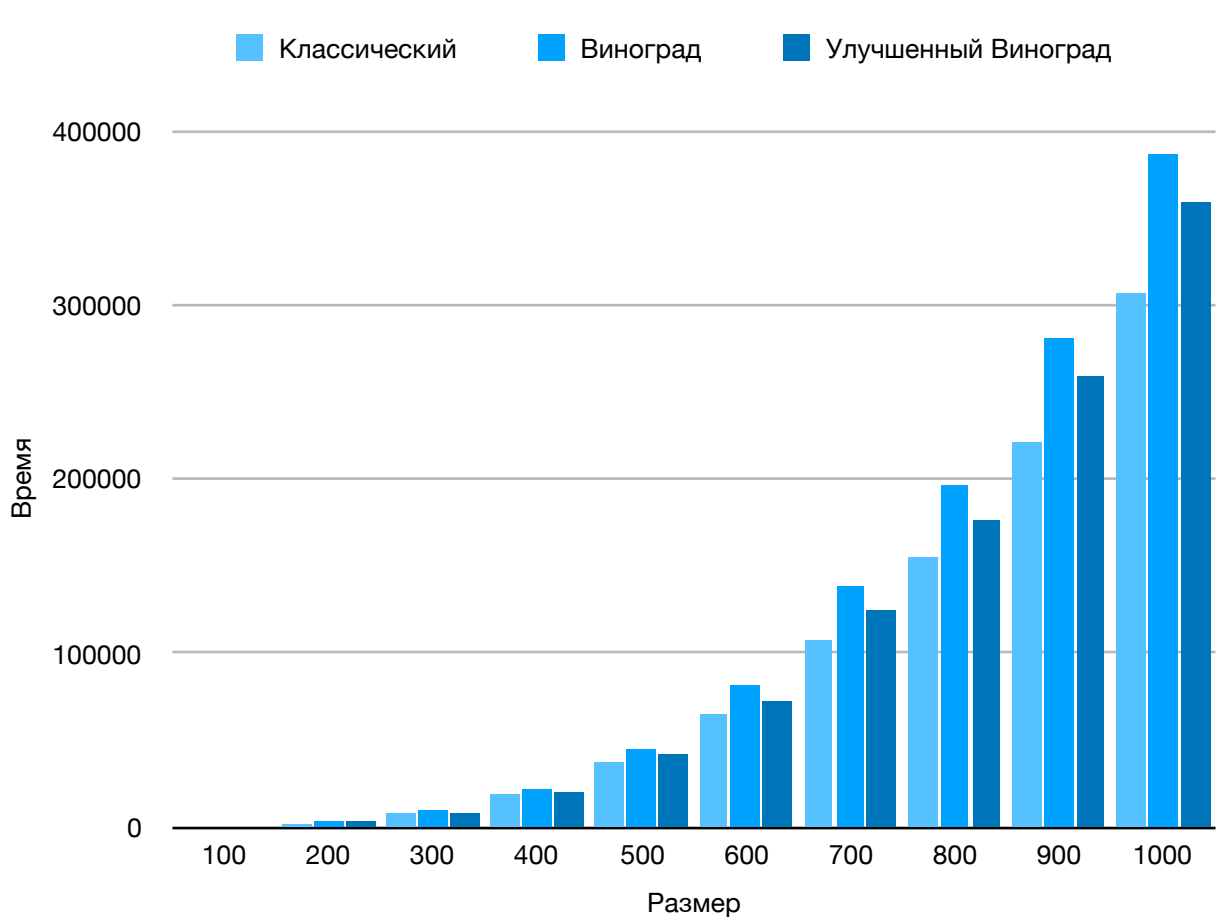

Рис. 4: График зависимости времени от размера (четный размер матрицы)

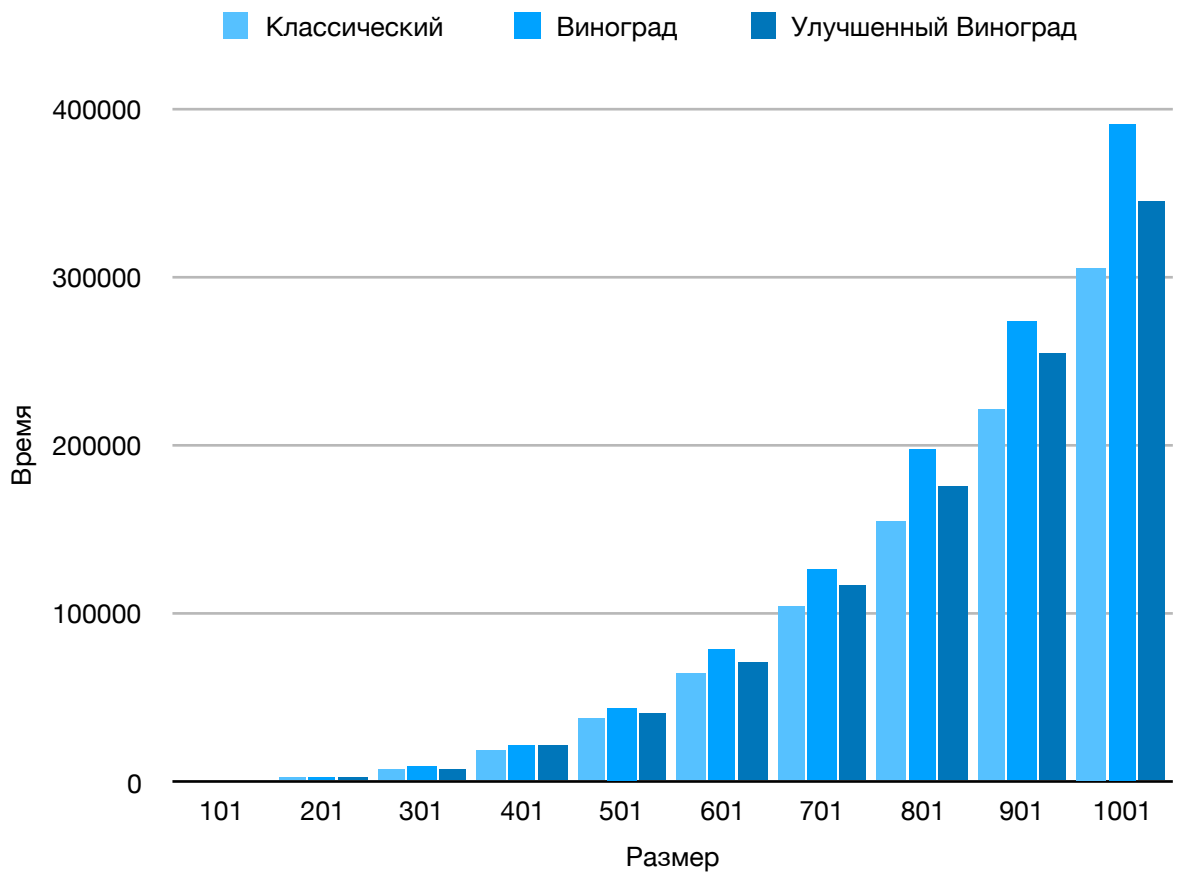

Рис. 5: График зависимости времени от размера (нечетный размер матрицы)

В результате проведенных испытаний алгоритмов было установлено, что:

1. Алгоритм Винограда начинает выигрывать в быстродействии у других известных алгоритмов только для матриц, размер которых превышает память современных компьютеров.

2. В классическом алгоритме разница времени между выполнением умножения матриц размером, отличающимся на единицу, незначительна, тогда как в алгоритме Винограда разница больше из-за дополнительной проверки на нечетное кол-во элементов

Заключение

В ходе лабораторной работы был реализован и улучшен алгоритм Винограда. Были получены навыки оптимизации кода на python.