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»

Щербатюк Дарья, ИУ7-54

Москва, 2017

Оглавление

Постановка задачи

Реализовать алгоритмы умножения матриц:

1. классический алгоритм умножения;
2. алгоритм Винограда;
3. улучшенный алгоритм Винограда.

Рассчитать сложность алгоритмов и провести временные эксперименты

Листинг

CLASSIC_MULTI_MATRIX:

```
def classic_multi(A, B):
        if len(B) != len(A[0]):
                print("Different dimension of the matrics")
                return

        n = len(A)
        m = len(A[0])
        t = len(B[0])

        answer = [[0 for i in range(t)] for j in range(n)]
        for i in range(n):
                for j in range(m):
                        for k in range(t):
                                answer[i][k] += A[i][j] * B[
                                    j][k]
        return answer
```

IMPRV_CLASSIC_MULTI_MATRIX:

```
def imprv_classic_multi(A, B):
        if len(B) != len(A[0]):
                print("Different dimension of the matrics")
                return
        return [[sum(x * B[i][col] for i, x in enumerate(row
            )) for col in range(len(B[0]))] for row in A]
```

WINOGRAD_MULTI_MATRIX:

```
def winograd_multi(G, H):
        a = len(G)
        b = len(H)
        c = len(H[0])

        if b != len(G[0]):
                print("Different dimension of the matrics")
                return
        d = b // 2
```

```
        row_factor = [0 for i in range(a)]
        col_factor = [0 for i in range(c)]

        # Row Factor calc
        for i in range(a):
            for j in range(d):
                row_factor[i] += G[i][2 * j] * G[i][2 * j + 1]

        # Col Factor calc
        for i in range(c):
            for j in range(d):
                col_factor[i] += H[2 * j][i] * H[2 * j + 1][i]

        answer = [[0 for i in range(c)] for j in range(a)]
        for i in range(a):
            for j in range(c):
                answer[i][j] = - row_factor[i] - col_factor[j]
                for k in range(d):
                    answer[i][j] += ((G[i][2 * k] + H[2 * k + 1][j]) * (G[i][2 * k + 1] + H[2 * k][j]))

        # For odd matrix
        if b % 2:
            for i in range(a):
                for j in range(c):
                    answer[i][j] += G[i][b - 1] * H[b - 1][j]

        return answer
```

IMPRV_WINOGRAD_MULTI_MATRIX:

```
    def winograd_multi(G, H):
        a = len(G)
        b = len(H)
        c = len(H[0])

        if b != len(G[0]):
            print("Different dimension of the matrics")
            return

        d = b // 2

        row_factor = [0 for i in range(a)]
        col_factor = [0 for i in range(c)]

```

```
        # Row Factor calculation
        for i in range(a):
            row_factor[i] = sum(G[i][2 * j] * G[i][2 * j + 1] for j in range(d))

        # Column Factor calculation
        for i in range(c):
            col_factor[i] = sum(H[2 * j][i] * H[2 * j + 1][i] for j in range(d))

        answer = [[0 for i in range(c)] for j in range(a)]
        for i in range(a):
            for j in range(c):
                answer[i][j] = sum((G[i][2 * k] + H[2 * k + 1][j]) * (G[i][2 * k + 1] + H[2 * k][j]) for k in range(d)) - row_factor[i] - col_factor[j]


        # For odd matrix
        if b % 2:
            for i in range(a):
                answer[i][j] = sum(G[i][b - 1] * H[b - 1][j] for j in range(c))

        return answer
```

Модель вычислений

Введём модель вычисления, используемую при оценках трудоёмкости:

1. вызов метода объекта класса имеет трудоёмкость 1;
2. объявление переменной/массива/структуры без определения имеет трудоёмкость 0;
3. операторы +, -, *, /, =, а также ++ и – имеют трудоёмкость 1;
4. условный оператор (без условий внутри) имеет трудоёмкость 0;
5. логические операции имеют трудоёмкость 1;
6. оператор цикла имеет трудоёмкость 1 + n(3 + T), где n – это число повторений цикла, T – трудоёмкость тела цикла;
7. одно присваивание до цикла (1 операция), внутри цикла присваивание, сравнение и инкремент (3 операции).
8. вызов функции имеет трудоёмкость 0, так как функции, объявленные внутри класса, компилятор рассматривает как inline и подставляет сразу вместо вызова код.

9. Fn(n) – часть трудоёмкости, зависящая только от размера входа;

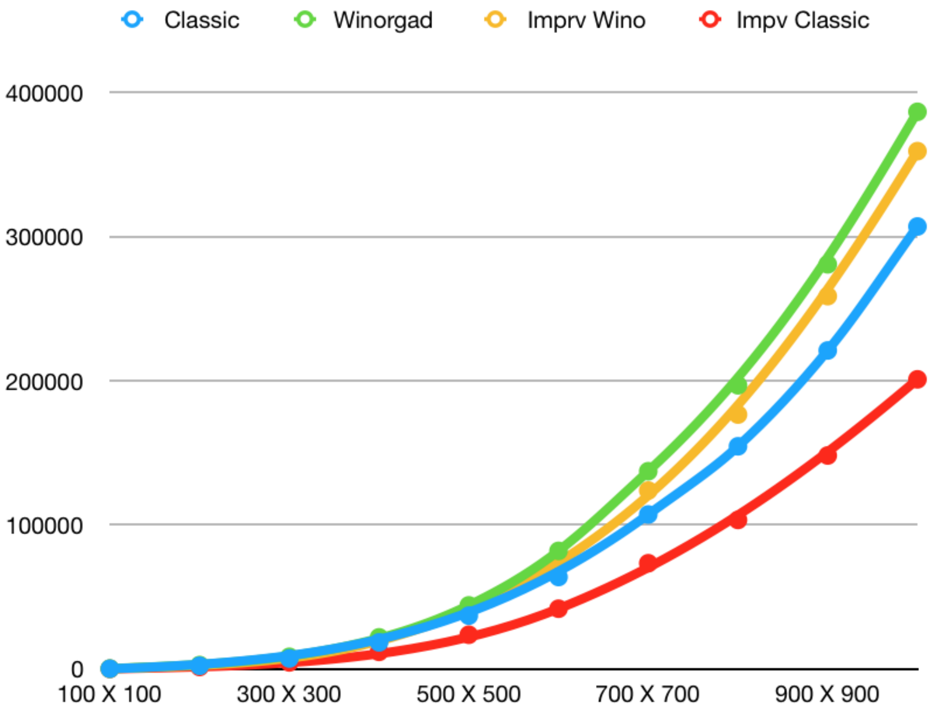
10. P – часть трудоёмкости, зависящая от конкретного входа, значений переменных.

Временные эксперименты

Измерения проводились для квадратных целочисленных матриц с помощью функиции clock() из встроенного модуля python time.

Size	Classic	Winorgad	Imprv Wino	Impv Classic
100 X 100	290.94333	361.24867	307.45867	186.74667
200 X 200	2353.78200	2815.53167	2542.62967	1510.97900
300 X 300	7412.31867	8559.82133	7807.55367	4662.86333
400 X 400	18560.03933	22043.19033	19563.29733	11975.96067
500 X 500	37094.78200	44213.58000	41490.46667	23845.95133
600 X 600	63905.64467	82005.13800	72646.38367	41914.59400
700 X 700	107256.68167	137257.87933	123870.87800	73450.65300
800 X 800	154546.39333	196856.36467	176563.80400	103469.35833
900 X 900	221015.69867	280759.76167	258684.84033	148171.84433
1000 X 1000	306991.44867	386513.90867	359238.27967	200993.01133
101 X 101	303.88400	356.69100	312.80333	186.54467
201 X 201	2230.60500	2600.42033	2351.83467	1424.86367
301 X 301	7662.25700	8809.80667	8061.61600	4859.16000
401 X 401	18658.00100	22335.59000	21140.32800	12475.44567
501 X 501	37049.78233	43786.63667	40248.24167	23620.58367
601 X 601	64789.63533	78651.74167	71082.76400	42363.01100
701 X 701	103867.78933	125626.53600	117352.42733	70031.39067
801 X 801	155306.45600	198218.44867	175573.72567	100056.82200
901 X 901	220950.60200	274289.26800	254960.42733	146510.15733
1001 X 1001	305296.35733	390898.75233	345251.03733	201508.00833

Замеры времени в миллисекундах (среднее из 5 замеров). Замер производился функцией clock из python-модуля time, которое изменяет затраченное время на процессоре в секундах



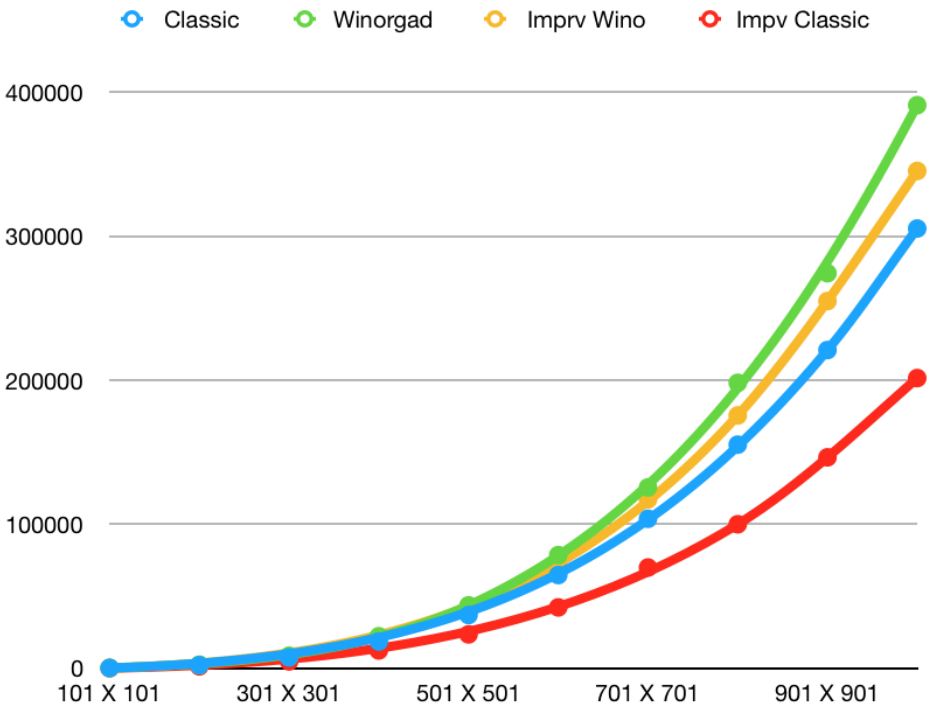


Рис. 1: Графики временных экспериментов

Рассчет сложности алгоритмов

Алгоритм Винограда:

1. $F_{row\_factor} = 2 + n(2 + 2 + d(2 + 10)) = 12nd + 4n + 2 = 6n + 4n + 2 = 10n + 2$
2. $F_{col\_factor} = 2 + m(2 + 2 + d(2 + 10)) = 12md + 4n + 2 = 6m + 4n + 2 = 10m + 2$
3. Вычисление матрицы :

$2+n(2+2+m(2+6+2+19d)) = 19dnm+10nm+4n+2 = 19/2(nm)+10nm+4n+2$

4. Вычисление последнего столбца (худший случай) :
$1+2+n(2+2+10m) = 10nm+4n+3$

Итого:

1. Лучший случай :
$19dnm+12dn+12dm+8n+4m+10n+6 = 19/2nm+24n+10m+6 \sim O(n^3)$

2. Худший случай :
$19dnm+12dn+12dm+20nm+4m+12n+9 = 59/2nm+18n+10m+9 \sim O(n^3)$

Улучшенный Алгоритм Винограда:

1. Рассчет $row\_factor : 2+n(2+(n-1)(2+8)) = 10n^2-8n+2$

2. Рассчет $col\_factor : 2+m(2+(n-1)(2+8)) = 10nm-8m+2$

3. Вычисление матрицы :
$2+n(2+m(2+6+2+(n-1)(2+14)+3)) = 2+2n-3nm+16n^2m$

4. Вычисление последнего столбца (худший случай) :
$1+2+n(2+2+m10) = 10nm+4n+3$

Итого:

1. Лучший случай : $6-6n+10n^2-8m+7nm+16n^2m \sim O(n^3)$

2. Худший случай : $9-2n+10n^2-8m+17nm+16n^2m \sim O(n^3)$

Классический алгоритм и улучшенный Классический алгоритм:

1. Вычисление матрицы : $2+n(2+2+t(2+2+9m)) = 9mnt+4nt+4n+2 \sim O(n^3)$

Выводы

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 были реализованы и улучшены 2 алгоритма умножения матриц: классический и алгоритм Винограда. Были получены навыки отпимизации кода на python, а так же работа с LaTeX. Изучен подход к вычислению сложности алгоритмов.