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:

```
1         def classic_multi(A, B):
2                 if len(B) != len(A[0]):
3                         print("Different dimension of the matrics")
4                         return
5
6                 n = len(A)
7                 m = len(A[0])
8                 t = len(B[0])
9
10                answer = [[0 for i in range(t)] for j in range(n)]
11                for i in range(n):
12                        for j in range(m):
13                                for k in range(t):
14                                        answer[i][k] += A[i][j] * B[
                                            j][k]
15                return answer
```

IMPRV_CLASSIC_MULTI_MATRIX:

```
1         def imprv_classic_multi(A, B):
2                 if len(B) != len(A[0]):
3                         print("Different dimension of the matrics")
4                         return
5                 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:

```
1         def winograd_multi(G, H):
2                 a = len(G)
3                 b = len(H)
4                 c = len(H[0])
5
6                 if b != len(G[0]):
7                         print("Different dimension of the matrics")
8                         return
9                 d = b // 2
```

```
10                row_factor = [0 for i in range(a)]
11                col_factor = [0 for i in range(c)]
12
13                # Row Factor calc
14                for i in range(a):
15                        for j in range(d):
16                                row_factor[i] += G[i][2 * j] * G[i
                                    ][2 * j + 1]
17
18                # Col Factor calc
19                for i in range(c):
20                        for j in range(d):
21                                col_factor[i] += H[2 * j][i] * H[2 *
                                     j + 1][i]
22
23                answer = [[0 for i in range(c)] for j in range(a)]
24                for i in range(a):
25                        for j in range(c):
26                                answer[i][j]  = - row_factor[i] -
                                    col_factor[j]
27                                for k in range(d):
28                                        answer[i][j]  += ((G[i][2 *
                                            k] + H[2 * k + 1][j]) * (
                                            G[i][2 * k + 1] + H[2 * k
                                            ][j]))
29
30                # For odd matrix
31                 if b % 2:
32                        for i in range(a):
33                                for j in range(c):
34                                        answer[i][j] += G[i][b - 1]
                                            * H[b - 1][j]
35
36                return answer
```

IMPRV_WINOGRAD_MULTI_MATRIX:

```
1         def winograd_multi(G, H):
2                 a = len(G)
3                 b = len(H)
4                 c = len(H[0])
5
6                 if b != len(G[0]):
7                         print("Different dimension of the matrics")
8                         return
9
10                d = b // 2
11
12                row_factor = [0 for i in range(a)]
13                col_factor = [0 for i in range(c)]
14
```

```
15                  # Row Factor calculation
16                  for i in range(a):
17                          row_factor[i] = sum(G[i][2 * j] * G[i][2 * j
                                + 1] for j in range(d))
18
19                  # Column Factor calculation
20                  for i in range(c):
21                          col_factor[i] = sum(H[2 * j][i] * H[2 * j +
                               1][i] for j in range(d))
22
23                  answer = [[0 for i in range(c)] for j in range(a)]
24                  for i in range(a):
25                          for j in range(c):
26                                  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]
27
28
29                  # For odd matrix
30                  if b % 2:
31                          for i in range(a):
32                                  answer[i][j] = sum(G[i][b - 1] * H[b
                                        - 1][j] for j in range(c))
33
34                  return answer
```

Временные эксперименты

Измерения проводились для квадратных целочисленных матриц

Size	Classic	Winorgad	Imprv Wino	Impv Classic
100 X 100	289.29400	347.95396	308.35764	185.55133
200 X 200	2249.77104	2667.67335	2386.45252	1452.55574
300 X 300	7694.16459	9144.59101	8145.12467	4935.43498
400 X 400	19090.08535	23198.14332	20572.56937	12945.18161
500 X 500	39005.07371	49364.15362	44845.09722	25241.92643
600 X 600	65456.91530	81830.84361	74260.43487	41960.06060
700 X 700	102072.69907	129947.46137	112226.29603	65716.18597
800 X 800	153583.53043	212627.15220	175107.29798	98817.42128
900 X 900	221113.73512	330751.41374	245380.58599	232635.11229
1000 X 1000	406853.16269	11982363.83335	388187.53799	199652.61801
101 X 101	326.57305	333.71878	335.29170	204.09226
201 X 201	2392.49770	2521.40872	2550.49467	1565.36229
301 X 301	8099.13405	8669.51084	8660.02146	5329.34825
401 X 401	19334.84364	20847.18029	21109.95770	12768.70171
501 X 501	39099.78644	42218.83734	42990.08997	25950.81258
601 X 601	66953.78153	76705.70691	73623.31589	44612.77151
701 X 701	106413.49713	118806.99031	119424.25927	70488.59978
801 X 801	158038.50412	184507.87044	176529.40289	103068.77263
901 X 901	223332.16429	4727406.52212	280284.77335	151464.52729
1001 X 1001	301034.56863	339312.07355	340042.95230	197018.43810

Замеры времени в миллисекундах (среднее из 5 замеров)

Рассчет сложности алгоритмов

Алгоритм Винограда:

1. Рассчет *row_factor* : $2 + n * (2 + 2 + d * (2 + 10)) = 12 * n * d + 4 * n + 2$
2. Рассчет *col_factor* : $2 + q * (2 + 2 + d * (2 + 10)) = 12 * q * d + 4 * n + 2$
3. Вычисление матрицы :
   $2 + n * (2 + 2 + q * (2 + 6 + 2 + d * 19)) = 19 * d * n * q + 10 * n * q + 4 * n + 2$
4. Вычисление последнего столбца (худший случай) :
   $1 + 2 + n * (2 + 2 + q * 10) = 10 * n * q + 4 * n + 3$

Итого:

1. Лучший случай : $19 * d * n * q + 12 * d * n + 12 * d * q + 8 * n + 4 * q + 10 * n + 6 \sim O(n^3)$
2. Худший случай :
   $19 * d * n * q + 12 * d * n + 12 * d * q + 20 * n * q + 4 * q + 12 * n + 9 \sim O(n^3)$

Улучшенный Алгоритм Винограда:

1. Рассчет *row_factor* : $2 + n * (2 + (n - 1) * (2 + 8)) = 10 * n^2 - 8 * n + 2$
2. Рассчет *col_factor* : $2 + q * (2 + (n - 1) * (2 + 8)) = 10 * n * q - 8 * q + 2$
3. Вычисление матрицы :
   $2 + n * (2 + q * (2 + 6 + 2 + (n - 1) * (2 + 14) + 3)) = 2 + 2 * n - 3 * n * q + 16 * n^2 * q$
4. Вычисление последнего столбца (худший случай) :
   $1 + 2 + n * (2 + 2 + q * 10) = 10 * n * q + 4 * n + 3$

Итого:

1. Лучший случай : $6 - 6 * n + 10 * n^2 - 8 * q + 7 * n * q + 16 * n^2 * q \sim O(n^3)$
2. Худший случай : $9 - 2 * n + 10 * n^2 - 8 * q + 17 * n * q + 16 * n^2 * q \sim O(n^3)$

Классический алгоритм и улучшенный Классический алгоритм:

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

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.