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»

Отчет по лабораторной работе № 4 по дисциплине "Анализ алгоритмов"

Тема Многопоточное программирование

Студент Хамзина Р. Р.

Группа ИУ7-53Б

Оценка (баллы) _____

Преподаватель Волкова Л. Л.

Москва — 2021 г.

Содержание

Введение

Одной из задач программирования является ускорение решения вычислительных задач. Один из способов ее решения - использование параллельных вычислений.

В последовательном алгоритме решения какой-либо задачи есть операции, которые может выполнять только один процесс, например, операции ввода и вывода. Кроме того, в алгоритме могут быть операции, которые могут выполняться параллельно разными процессами. Алгоритм, операции которого могут быть выполнены разными процессами параллельно, называют параллельным. Каждый процесс состоит из одного или нескольких потоков. Свойство, состоящее в разделении процесса на потоки, выполняющие задачи параллельно, называют многопоточностью.

Примером вычислительных задач являются алгоритмы обработки графов. Графом называют конечное множество вершин и множество ребер. Каждому ребру сопоставлены две вершины - концы ребра. Число вершин графа называют порядком. Для распараллеливания может быть рассмотрена задача поиска кратчайших путей между всеми парами вершин графа. Данная задача решается при помощи алгоритма Флойда.

Целью данной лабораторной работы является изучение многопоточности на основе алгоритма Флойда поиска кратчайших расстояний между всеми парами вершин графа. Для достижения поставленной цели требуется выполнить следующие задачи:

- изучить основы многопоточности;
- изучить способ представления графа;
- изучить алгоритм поиска кратчайших расстояний между всеми парами вершин графа;
- привести схемы изучаемого алгоритма;
- описать используемые типы и структуры данных;
- описать структуру разрабатываемого программного обеспечения;

- определить средства программной реализации выбранного алгоритма;
- реализовать разработанный алгоритм;
- провести функциональное тестирование программного обеспечения;
- провести сравнительный анализ по времени реализованного алгоритма;
- подготовить отчет о выполненной лабораторной работе.

1 Аналитическая часть

В данном разделе будут описаны основы многопоточности, способ представления графа и алгоритм Флойда поиска кратчайших расстояний между всеми парами вершин графа.

1.1 Многопоточность

1.2 Представление графа

1.3 Алгоритм Флойда

1.4 Вывод

Были изучены основы распараллеливания алгоритмов, способ представления графа при помощи матрицы смежности и алгоритм Флойда поиска кратчайших путей между всеми парами вершин графа.

Программе, реализующей данный алгоритм, на вход будет подаваться матрица смежности, которая задает граф. Выходными данными такой программы должна быть матрица смежности кратчайших путей между всеми парами вершин графа. Программа должна работать в рамках следующих ограничений:

- веса ребер графа - целые неотрицательные числа;
- при отсутствии пути между вершинами обозначается -1;
- должно быть выдано сообщение об ошибке при вводе пустой матрицы смежности или при вводе недопустимого веса ребра графа.

Пользователь должен иметь возможность выбора построения алгоритма с распараллеливанием и без него. В случае параллельного алгоритма пользователь должен иметь возможность ввода числа потоков. Также

должны быть реализованы сравнение алгоритмов по времени работы в зависимости от числа потоков и в зависимости от порядка графа с выводом результатов на экран и получение графического представления результатов сравнения. Результат данных действий пользователь должен получать при помощи меню.

2 Конструкторская часть

В данном разделе будут представлены схемы алгоритмов умножения матриц: стандартного алгоритма, алгоритма Винограда и оптимизированного алгоритма Винограда. Будут описаны типы и структуры данных, используемые для реализации, а также структура разрабатываемого программного обеспечения. Кроме того, будут выделены классы эквивалентности для тестирования.

2.1 Разработка алгоритмов

На рисунке **??** приведена схема стандартного алгоритма умножения матриц. Схема умножения матриц по алгоритму Винограда приведена на рисунках **??-??**, схема оптимизированной версии приведена на рисунках **??-??**.

2.2 Модель вычислений для оценки трудоемкости алгоритмов

Для определения трудоемкости алгоритмов необходимо ввести модель вычислений [3]:

1. операции из списка (2.1) имеют трудоемкость равную 1;

$$+, -, /, *, \%, =, +=, -=, *=, /=, \%=, ==, !=, <, >, <=, >=, [], ++, -- \tag{2.1}$$

2. трудоемкость оператора выбора `if условие then A else B` рассчитывается, как (2.2);

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.2}$$

3. трудоемкость цикла рассчитывается, как (2.3);

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремент}} + f_{\text{сравнения}}) \quad (2.3)$$

4. трудоемкость вызова функции равна 0.

2.3 Трудоемкость алгоритмов

Проведем сравнительный анализ реализованных алгоритмов по трудоемкости для умножения матрицы A[N][P] и матрицы B[P][M].

2.3.1 Стандартный алгоритм умножения матриц

Трудоемкость данного алгоритма будет складываться из:

- трудоемкости цикла по $k \in [0..P-1]$, равной $1 + P(3+8) = 1 + 11P$;
- трудоекмости цикла по $j \in [0..M-1]$, равной $1 + M(3 + f_{body}) = 1 + M(3 + 1 + 11P) = 1 + 4M + 11PM$;
- трудоемкости цикла по $i \in [0..N-1]$, равной $1 + N(3 + f_{body}) = 1 + N(3 + 1 + 4M + 11MP) = 1 + 4N + 4NM + 11NMP$.

Таким образом, трудоемкость стандартного алгоритма умножения равна $f_{standard} = 1 + 4N + 4NM + 11NMP \approx 11NMP$.

2.3.2 Алгоритм умножения матриц по Винограду

Трудоемкость данного алгоритма будет складываться из:

- трудоемкости создания и заполнения нулями векторов row_factors и column_factors, равной $N + M$;
- трудоемкости предварительных вычислений для строк, равной $1 + N(3 + 1 + P/2(3 + 12)) = 1 + 4N + 7.5NP$;

- трудоемкости предварительных вычислений для столбцов, равной $1+M(3+1+P/2(3+12))=1+4M+7.5MP$;
- трудоемкости цикла по $k \in [0..P/2-1]$, равной $1+P/2(3+23)=1+13P$;
- трудоекмости цикла по $j \in [0..M-1]$, равной $1+M(3+f_{body})=1+M(3+7+1+13P)=1+11M+13PM$;
- трудоемкости цикла по $i \in [0..N-1]$, равной $1+N(3+f_{body})=1+N(3+1+11M+13MP)=1+4N+11NM+13NMP$;
- трудоемкости условия, равной 2;
- трудоемкости двойного цикла добавления в случае нечетного размера матрицы, равной $1+N(3+1+M(3+13))=1+4N+16NM$.

Таким образом, трудоемкость алгоритма умножения по Винограду в худшем случае (нечетный размер матрицы) равна $f_{worst}=N+M+2(1+4N+7.5M)+1+4N+11NM+13NMP+2+1+4N+16NM=6+17N+16M+27NM+13NMP \approx 13NMP$.

Трудоемкость алгоритма умножения по Винограду в лучшем случае (четный размер матрицы) равна $f_{best}=N+M+2(1+4N+7.5M)+1+4N+11NM+13NMP+2=5+13N+16M+11NM+13NMP \approx 13NMP$.

2.3.3 Оптимизированный алгоритм умножения матриц по Винограду

Трудоемкость данного алгоритма будет складываться из:

- трудоемкости создания и заполнения нулями векторов row_factors и column_factors, равной $N+M$;
- трудоемкости предварительных вычислений для строк, равной $1+N(3+1+P/2(3+8))=1+4N+5.5NP$;
- трудоемкости предварительных вычислений для столбцов, равной $1+M(3+1+P/2(3+8))=1+4M+5.5MP$;

- трудоемкости вычисления признака четности размера матрицы, равной 2;
- трудоемкости цикла по $k \in [0..P/2-1]$, равной $1 + P/2(3+16) = 1 + 9.5P$;
- трудоемкости условия, равной 1;
- трудоемкости добавления в случае нечетного размера матрицы, равной 10;
- трудоекмости цикла по $j \in [0..M-1]$, равной $1 + M(3 + f_{body})$;
- трудоемкости цикла по $i \in [0..N-1]$, равной $1 + N(3 + f_{body})$.

Таким образом, трудоемкость оптимизированного алгоритма умножения по Винограду в худшем случае (нечетный размер матрицы) равна $f_{worst} = N + M + 2(1 + 4N + 5.5M) + 2 + 1 + N(3 + 1 + M(3 + 7 + 1 + 9.5P + 1 + 10)) = 5 + 13N + 12M + 22MN + 9.5MNP \approx 9.5NMP$.

Трудоемкость оптимизированного алгоритма умножения по Винограду в лучшем случае (четный размер матрицы) равна $f_{best} = N + M + 2(1 + 4N + 5.5M) + 2 + 1 + N(3 + 1 + M(3 + 7 + 1 + 9.5P + 1)) = 5 + 13N + 12M + 11MN + 9.5MNP \approx 9.5NMP$.

2.4 Описание используемых типов и структур данных

Для реализации рассмотренных алгоритмов будет использован тип данных *int* - для числа строк и числа столбцов каждой матрицы.

Структура данных - матрица - представляет собой двумерный список значений типа *int*.

2.5 Структура разрабатываемого ПО

При реализации разрабатываемого программного обеспечения будет использоваться метод структурного программирования. Для взаимодействия

с пользователем будет выделена функция process(), из которой будут вызываться методы умножения матриц и функции сравнительного анализа. Для работы с матрицами будут разработаны следущие функции:

- создание матрицы случайных чисел, входным параметром функции является размер, выходным - матрица;
- ввод матрицы, функция не имеет входных параметров, выходным параметром является введенная матрица;
- процедура вывода матрицы, входным параметром которой является матрица;
- функции умножения матриц для каждого алгоритма (классического, Винограда и оптимизированного Винограда), у которых на входе - две матрицы для умножения, а на выходе - результат умножения матриц.

Для сравнительного анализа будут реализованы:

- процедура замеров времени, выходным парметром которой является массив временных значений;
- функция графического представления замеров времени, у которой на входе - массив временных значений, на выходе - его графическое представление.

2.6 Классы эквивалентности при тестировании

Для тестирования разрабатываемой программы будут выделены следующие классы эквивалентности:

- две пустые матрицы;
- одна пустая матрица, одна нет;
- число столбцов первой матрицы не равно числу строк второй;

- умножение квадратных матриц;
- умножение матриц с одинаковым числом стоблцов первой и числом строк второй, но разным числом строк первой и числом столбцов второй.

2.7 Вывод

Были представлены схемы алгоритмов умножения матриц. Были указаны типы и структуры данных, используемые для реализации, и описана структура разрабатываемого программного обеспечения. Также были выделены классы эквивалентности для тестирования ПО.

3 Технологическая часть

В данном разделе будут указаны средства реализации, будут представлены листинги кода, а также функциональные тесты.

3.1 Средства реализации

Реализация данной лабораторной работы выполнялась при помощи языка программирования Python [4]. Выбор ЯП обусловлен простотой синтаксиса, большим числом библиотек и эффективностью визуализации данных.

Замеры времени проводились при помощи функции process_time из библиотеки time [5].

3.2 Сведения о модулях программы

Программа состоит из следующих модулей:

- main.py - главный файл программы, предоставляющий пользователю меню для выполнения основных функций;
- matrix.py - файл, содержащий функции работы с матрицами;
- time_test.py - файл, содержащий функции замеров времени работы указанных алгоритмов;
- graph_result.py - файл, содержащий функции визуализации временных характеристик описанных алгоритмов.

3.3 Листинги кода

3.4 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для функций, реализующих алгоритмы умножения матриц. Все тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Матрица A	Матрица B	Ожидаемый результат
$\begin{pmatrix} \end{pmatrix}$	$\begin{pmatrix} \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} 1 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} -1 & 0 & -1 \end{pmatrix}$	Сообщение об ошибке
$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$	$\begin{pmatrix} -1 & -2 & -3 \\ -4 & -5 & -6 \\ -7 & -8 & -9 \end{pmatrix}$	$\begin{pmatrix} -30 & -36 & -42 \\ -66 & -81 & -96 \\ -102 & -126 & -150 \end{pmatrix}$
$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$	$\begin{pmatrix} 14 \end{pmatrix}$

3.5 Вывод

Были реализованы функции алгоритмов умножения матриц. Было проведено функциональное тестирование указанных функций.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программы, и будет проведен сравнительный анализ реализованных алгоритмов умножения матриц по затраченному процессорному времени.

4.1 Технические характеристики

Тестирование проводилось на устройстве со следующими техническими характеристиками:

- операционная система: Ubuntu 20.04.1 Linux x86_64 [6];
- память : 8 GiB;
- процессор: AMD® Ryzen™ 3 3200u @ 2.6 GHz [7].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.2 Демонстрация работы программы

На рисунке 4.1 приведен пример работы программы.

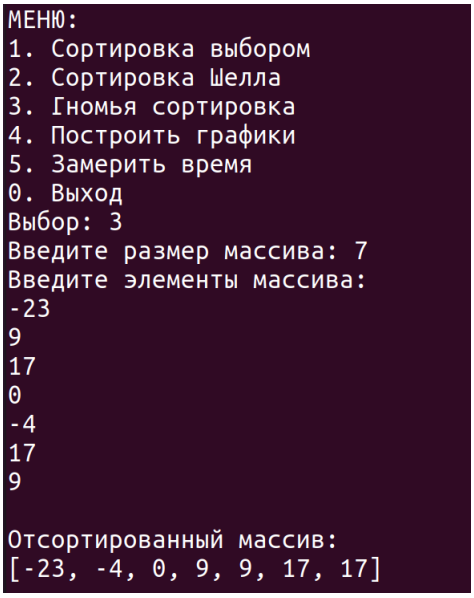

Рисунок 4.1 – Пример работы программы

4.3 Время выполнения алгоритмов

Функция process_time из библиотеки time ЯП Python возвращает процессорное время в секундах - значение типа float.

Для замера времени:

- получить значение времени до начала выполнения алгоритма, затем после её окончания. Чтобы получить результат, необходимо вычесть из второго значения первое;
- первый шаг необходимо повторить iters раз (в программе iters равно 10), суммируя полученные значения, а затем усреднить результат.

Замеры проводились для квадратных матриц целых чисел, заполненных случайным образом, размером от 10 до 100 и от 11 до 101. Результаты измерения времени для четного размера матриц приведены в таблице 4.1 (в мс).

Таблица 4.1 – Результаты замеров времени

Размер	Стандартный	Винограда	Опт-ый Винограда
10	0.2875	0.6851	0.2061
20	1.6220	3.5965	1.2381
30	5.3519	4.4189	4.0736
40	12.2201	9.7817	9.0212
50	23.4212	18.8085	18.8640
60	39.3355	32.1766	30.2312
70	61.6586	52.5354	50.6840
80	91.7189	81.2931	73.1704
90	129.2349	110.4791	104.2271
100	177.5043	158.5571	134.2171

На рисунке **??** приведены графические результаты сравнения временных характеристик для четного размера матриц.

Результаты измерения времени для четного размера матриц приведены в таблице 4.2 (в мс).

Таблица 4.2 – Результаты замеров времени

Размер	Стандартный	Винограда	Опт-ый Винограда
11	0.2942	0.8884	0.2907
21	1.8007	3.2802	1.6107
31	5.5211	4.6575	4.8589
41	12.0378	10.6847	10.6740
51	23.3503	20.2865	19.9010
61	41.0413	34.2695	32.9210
71	65.2536	54.6966	50.5432
81	93.8148	82.2679	72.8342
91	131.0258	118.6902	105.8783
101	177.7476	158.8673	142.5004

На рисунке **??** приведены графические результаты сравнения временных характеристик для нечетного размера матриц.

4.4 Вывод

В результате эксперимента было получено, что при размере матриц, большем 30, оптимизированный алгоритм Винограда работает работает быстрее стандартного алгоритма в 1.3 раза. При этом стандартный алгоритм медленнее алгоритма Винограда в 1.2 раза. Тогда, для размера матриц, начиная с 30 элементов, небходимо использовать оптимизированный алгоритм умножения матриц по Винограду.

Также в результате эксперимента было установлено, что при четном размере матриц, алгоритм Винограда работает быстрее, чем на матрицах с нечетным размером в 1.2 раза в связи с проведением дополнительных вычислений для крайних строк и столбцов. Можно сделать вывод, что алгоритм Винограда предпочтительно использовать для умножения матриц четных размеров.

Заключение

В результате исследования было получено, что при размере матриц, большем 30, небходимо использовать оптимизированный алгоритм умножения матриц по Винограду, так как данный алгоритм работает быстрее стандартного алгоритма в 1.3 раза. При этом стандартный алгоритм медленнее алгоритма Винограда в 1.2 раза.

Кроме того алгоритм Винограда предпочтительно использовать для умножения матриц четных размеров, так как указанный алгоритм работает в 1.2 раза быстрее, чем на матрицах с нечетным размером. Это связано с проведением дополнительных вычислений для крайних строк и столбцов.

Цель, поставленная перед началом работы, была достигнута. В ходе лабораторной работы были решены следующие задачи:

- были изучены классический алгоритм, алгоритм Винограда и его оптимизированная версия умножения матриц;
- были разработаны изученные алгоритмы;
- был проведен сравнительный анализ реализованных алгоритмов;
- был подготовлен отчет о выполненной лабораторной работе.

Список литературы

[1] Н.Вирт Алгоритмы и структуры данных. 1989.

[2] Д. А. Погорелов, А. М. Таразанов Оптимизация классического алгоритма Винограда для перемножения матриц. 2019.

[3] М. В. Ульянов Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ. 2007.

[4] Welcome to Python [Электронный ресурс]. Режим доступа: `https://www.python.org` (дата обращения: 18.10.2021).

[5] time — Time access and conversions [Электронный ресурс]. Режим доступа: `https://docs.python.org/3/library/time.html#functions` (дата обращения: 18.10.2021).

[6] Ubuntu 20.04 LTS (Focal Fossa) Beta [Электронный ресурс]. Режим доступа: `http://old-releases.ubuntu.com/releases/20.04.1/` (дата обращения: 18.10.2021).

[7] Мобильный процессор AMD Ryzen™ 3 3200U с графикой Radeon™ Vega 3 [Электронный ресурс]. Режим доступа: `https://www.amd.com/ru/products/apu/amd-ryzen-3-3200u` (дата обращения: 18.10.2021).