МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра информационной безопасности

ПОСТАНОВКА ЗАДАЧИ

по лабораторным работам №№ 2, 3

по дисциплине «Алгоритмы и структуры данных»

Тема: Экспериментальное нахождение сложности алгоритма

Преподаватель _____ Абросимов И.К.

Санкт-Петербург
2025

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТ И ПОСТАНОВКА ЗАДАЧ

Цель работы: экспериментальное нахождение средней сложности предложенного алгоритма по его реализации на указанном языке программирования.

В лабораторной работе № 2 экспериментально определяется средняя временная сложность, в лабораторной работе № 3 – средняя пространственная сложность.

Задание на лабораторную работу:

1) Реализовать и отладить на языке C или C++ структуру данных и операции над ней согласно варианту;

2) Реализовать алгоритм согласно варианту с использованием реализованной структуры данных;

3) Выполнить экспериментальное нахождение средней сложности реализации исследуемого алгоритма.

Содержание пояснительной записки: разделы «СОДЕРЖАНИЕ», «ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ» с подразделами «Описание исследуемого алгоритма» и «Методика экспериментального исследования», «РЕШЕНИЕ ПОСТАВЛЕННОЙ ЗАДАЧИ» с подразделами «Реализация структуры данных», «Реализация исследуемого алгоритма», «Тестирование реализации алгоритма», «Проведение эксперимента» и «Обработка результатов эксперимента», «ЗАКЛЮЧЕНИЕ», «СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ».

2 МЕТОДИКИ ЭКСПЕРИМЕНТАЛЬНОГО ИССЛЕДОВАНИЯ

2.1 Определение средней временной сложности

В работе исследуется средняя временная сложность алгоритма – среднее количество временных затрат алгоритма A на множестве входных данных заданного размера n, обозначаемый как $\bar{T}_A(n)$

$$\bar{T}_A(n) = \frac{1}{|X|} \cdot \sum_{x \in X_n} C_A^T(x), \tag{2.1}$$

где $X_n = \{x \mid \|x\| = n\}$ – множество входных данных размера n, $C_A^T(x)$ – временные затраты алгоритма, равные количеству времени, которое затрачивается на выполнение алгоритма A при входных данных $x \in X_n$ размера $\|x\| = n$. В работе $C_A^T(x)$ измеряется непосредственно, например, с помощью функции `time()` из заголовочного файла `<ctime>`.

Нахождение функции $\bar{T}_A(n)$ в виде выражения с конкретными константами слишком трудоемко, поскольку требует аналитического построения распределения временных затрат входных данных из множества X_n. Поэтому вместо нахождения точного вида описывающей $\bar{T}_A(n)$ функции часто ограничиваются определением некоторого множества функций, к которому принадлежит $\bar{T}_A(n)$.

Для оценивания временной сложности в сверху используется символ «о большое». Говорят, что $f(n) \in O\big(g(n)\big)$ тогда и только тогда, когда существует положительное вещественное c и натуральное N такое, что для любого $n > N$ выполняется неравенство

$$|f(n)| \leq c \cdot |g(n)|. \tag{2.2}$$

Для оценивания алгоритмов обычно служат следующие функции: константа, логарифм, степень, произведение степени на логарифм, экспонента, факториал. При этом чем быстрее растут значения оцениваемой функции, тем больше времени потребует выполнение алгоритма.

Для определения класса функции, которому принадлежит временная сложность, обычно используются случайно выбранные наборы входных данных фиксированного размера n_1, n_2, …, n_k, где $k \geq 5$. Пусть t_1, t_2, …, t_k – измеренное время работы алгоритма при соответствующем размере входных данных, f – функция такая, что $T(n) \in O(f(n))$, тогда $t_i = T(n_i) = C \cdot f(n_i)$, где $C > 0$ – некоторая константа и функцию f можно определить следующим образом. Выберем размеры входных данных так, чтобы было справедливо приближенное равенство $\frac{t_{i+1}}{t_i} \approx \alpha$, где α – некоторая константа. Тогда, с учетом

$$\frac{t_{i+1}}{t_i} = \frac{C \cdot f(n_{i+1})}{C \cdot f(n_i)} = \frac{f(n_{i+1})}{f(n_i)}. \tag{2.3}$$

имеем $f(n_{i+1}) = \alpha \cdot f(n_i)$ при $i \in \overline{1, k-1}$, тогда значение данного отношения характеризует скорость роста функции f, по которой можно определить саму функцию f, как решение системы

$$\begin{cases} n_{i+1} = g(n_i), \\ f(n_{i+1}) = \alpha \cdot f(n_i), \end{cases} \tag{2.4}$$

причем общий вид функции f известен (например, степенная, показательная).

Если известно, что алгоритм выполняется за полиномиальное время, то размер входных данных после каждого измерения времени рекомендуется удваивать. Если же алгоритм выполняется за время, не меньшее экспоненциального, то размер входных данных рекомендуется увеличивать на небольшую константу. Таким образом, n_{i+1} и n_i обычно связаны функцией g одного из следующих видов

$$g(n_i) = 2n_i, \tag{2.5}$$

$$g(n_i) = n_i + m, \tag{2.6}$$

где m – фиксированная положительная константа.

Особый случай возникает, когда $\frac{t_{i+1}}{t_i} \neq const$ даже при выборе $n_{i+1} = n_i + 1$. В этом случае следует выполнить интерполяцию отношений $\frac{t_{i+1}}{t_i}$ для всех $i \in \overline{1, k-1}$. Пусть $\alpha(n)$ – результат такой интерполяции, тогда

$$f(n+1) = \alpha(n) \cdot f(n). \quad (2.7)$$

Это означает, что алгоритм имеет сложность, сравнимую с факториальной или даже превосходящую ее.

Размер входных данных при проведении экспериментов рекомендуется выбирать таким, чтобы, с одной стороны, они были достаточно «большими», так как оценивается функция $f(n)$ в предположении большого размера n входных данных, с другой стороны, не чрезмерно большими, чтобы время измерения было разумным (не более двадцати минут на все измерения времени).

В случае, когда время работы алгоритма настолько мало, что его нельзя измерить средствами используемого языка программирования, следует выполнять генерацию некоторого количества N входных данных фиксированного размера, измерять общее время t_i' выполнения алгоритма для всех сгенерированных входах, после чего вычислить время работы на одном входе как $t_i = \frac{t_i'}{N}$.

В случае, когда входных параметров у алгоритма несколько, среди них выбирается тот, который вносит наибольший вклад с точки зрения анализа сложности. Если такой выбор сделать не удается, то следует анализировать поведение каждого параметра отдельно, либо в совокупности (например, связав длины входных данных алгоритма при помощи некоторой функции).

2.2 Определение средней пространственной сложности

В работе исследуется средняя пространственная сложность алгоритма – среднее количество пространственных затрат алгоритма A на множестве входных данных заданного размера n, обозначаемый как $\bar{S}_A(n)$

$$\bar{S}_A(n) = \frac{1}{|X|} \cdot \sum_{x \in X_n} C_A^S(x), \tag{2.8}$$

где $X_n = \{x \mid \|x\| = n\}$ – множество входных данных размера n, $C_A^S(x)$ – пространственные затраты алгоритма, равные количеству времени, которое затрачивается на выполнение алгоритма A при входных данных $x \in X_n$ размера $\|x\| = n$.

Как и для временной сложности, чаще всего определяют класс функций, к которому принадлежит $\bar{S}_A(n)$ при $n \to \infty$, поэтому ее описывают с использованием символов асимптотических оценок, таких как «о большое». Поэтому экспериментальное оценивание средней пространственной сложности может быть проведено по той же методике, что и оценивание средней временной сложности, за исключением случая, когда средняя пространственная сложность описывается полиномиальной функцией, а средняя временная сложность – сверхполиномиальной (например, экспонентой или факториалом). В этом случае для определения функции, описывающей скорость роста пространственной сложности, вместо (2.3) используется, с учетом, что $n_{i+1} = n_i + m$ формула

$$n_i \cdot \frac{s_{i+1} - s_i}{s_i} = n_i \cdot \frac{C \cdot (n_{i+1}^a - n_i^a)}{C \cdot n_i^a} = \frac{(n_i + m)^a - n_i^a}{n_i^{a-1}} \xrightarrow[n_i \to \infty]{} am, \tag{2.9}$$

где s_i – используемая при работе алгоритма память при соответствующем размере входных данных.

Если данное отношение при всех $n_i, i \in \overline{1,5}$ примерно равно $a \cdot m$, то средняя пространственная сложность алгоритма, согласно (2.9), оценивается при помощи функции $O(n^a)$.

Подсчет количества памяти, которое потребляет программа, реализующая оцениваемый алгоритм, может быть произведен с помощью непосредственного подсчета с использованием дополнительных переменных, с использованием специальных функций API операционной систем, либо с использованием специального ПО, например, встроенного в среду разработки.

Рассмотрим способ непосредственного подсчета, причем учитывать будем только динамическую память. Определим две глобальные переменные: $currentMemory$ – количество динамической памяти, которое потребляется в текущий момент времени и $maxMemory$ – наибольшее количество динамической памяти, которое потреблялось в процессе программы и инициализируем их нулевыми значениями.

Если в какой-то момент $maxMemory < currentMemory,$ то $maxMemory$ присваивается значение $currentMemory$.

Перед каждым вызовом функции выделения динамической памяти $currentMemory$ увеличивается на величину, равную размеру выделенного блока памяти, а после каждого вызова функции освобождения динамической памяти – уменьшается на величину, равную размеру освобождаемого блока памяти.

2.3 Ход выполнения эксперимента и пример обработки результатов

Для выполнения эксперимента следует:

1) Выполнить измерения времени работы (потребляемой при работе памяти) рассматриваемого алгоритма при различных входных данных n_i.

2) Построить график измерений времени (памяти), через точку с наименьшей абсциссой провести «эталонный» график временной сложности (см. пример ниже);

3) Вывод должен содержать ответ на вопрос «К какому классу функций принадлежит средняя временная (пространственная) сложность и совпала ли она с теоретической оценкой сложности?»

Пример. Пусть было измерено время $t_1 = 36$ нс, $t_2 = 322$ нс, $t_3 = 2442$ нс, $t_4 = 19875$ нс, $t_5 = 158724$ нс для входных данных размера $n_1 = 40$ бит, $n_{i+1} = 2n_i, i \in \overline{1,4}$ (размер входных данных удваивался) и известно, что временная сложность алгоритма принадлежит классу степенных функций. Тогда, найдя отношения $\frac{t_2}{t_1} \approx 8{,}9$, $\frac{t_3}{t_2} \approx 7{,}6$, $\frac{t_4}{t_3} \approx 8{,}1$, $\frac{t_5}{t_4} \approx 8$ можно сделать вывод о том, что функция f удовлетворяет уравнению $f(2n) = 8f(n)$, откуда, с учетом, что $f(n) = n^a$ следует, что $f(2n) = (2n)^a = 2^a n^a = 8n^a$ и $a = 3$, т.е. алгоритм имеет кубическую сложность. «Эталонным» графиком временной сложности будет ломаная, построенная по точкам (n_i, y_i), где $i \in \overline{1,5}$, $y_{i+1} = 8y_i$, $y_1 = t_1$, экспериментальной кривой – ломаная, построенная по точкам (n_i, t_i).

3 ВАРИАНТЫ ЗАДАНИЙ И КОММЕНТАРИИ К НИМ

№	Название	Структура данных
1	операции над конечными неупорядоченными множествами	битовый массив
2	решето Эратосфена	
3	вычисление веса Хемминга	
4	бинарное деление «уголком»	длинное целое число
5	вычисление значения многочлена в точке	
6	произведение по модулю числа	
7	вычисление остатка без деления «уголком»	
8	быстрое возведение числа в степень	
9	вычисление символа Якоби	
10	умножение чисел через умножение многочленов	многочлен с коэффициентами из конечного кольца
11	деление «уголком» на унитарный многочлен	
12	произведение по модулю унитарного многочлена	
13	вычисление остатка без деления «уголком» на унитарный многочлен	
14	быстрое возведение квадратной матрицы в степень	матрица с коэффициентами из конечного кольца
15	алгоритм нахождения линейно зависимого набора строк двоичной матрицы	

Для выполнения лабораторных работ по оцениванию сложности заданного алгоритма используется одна из структур данных, основанных на динамических одномерных или двухмерных массивах целых чисел. При этом

для соответствующей структуры данных должны быть реализованы все перечисленные в таблице ниже операции, вне зависимости от того, используются ли рассматриваемыми алгоритмом все они, или только их часть.

№	Название структуры данных	Перечень операций
1	битовый массив	установление бита и группы подряд идущих бит, инверсия бита и группы подряд идущих бит, логический сдвиг влево и вправо, преобразование битового массива в строку типа `char*` и строки типа `char*` в битовый массив – всего 8 операций
2	длинное целое число	сравнение, сумма, разность, произведение на короткое, остаток от деления на короткое, частное от деления на короткое, преобразование длинного целого числа в строку типа `char*` и строки типа `char*` в длинное целое число – всего 8 операций
3	многочлен с коэффициентами из конечного кольца	сумма, разность, произведение на число, произведение на многочлен, стандартное скалярное произведение, остаток от деления на унитарный многочлен, преобразование многочлена в строку типа `char*` и строки типа `char*` в многочлен – всего 8 операций

№	Название структуры данных	Перечень операций
4	матрица с коэффициентами из конечного кольца	сумма, разность, произведение на число, транспонирование, произведение на матрицу согласованного размера, получение подматрицы (через левую, правую, верхнюю и нижнюю границу), преобразование матрицы в строку типа `char*` и строки типа `char*` в матрицу – всего 8 операций

Многочлен в виде строки типа `char*` представляет собой набор символов, описывающих коэффициенты, взятых в скобки и разделенных запятыми, например многочлену $3x^2 + 2x + 1$ соответствует строка

`"(1,2,3)"`

Матрица в виде строки типа `char*` представляет собой набор символов, описывающих элементы матрицы, взятых в скобки, причем соседние элементы в строке отделяются запятыми, а элементы соседних строк – точкой с запятой, например, матрице $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ соответствует строка

`"(1,2,3;4,5,6)"`

Остаток от деления на унитарный многочлен M вычисляется посредством использования формулы

$$x^{\deg(M)} \bmod M = x^{\deg(M)} - M. \qquad (3.1)$$

при этом, если требуется вычислить $x^{\deg(M)+1}$, то правая часть формулы умножается на x, после чего, при необходимости, применяется формула (3.1), для $x^{\deg(M)+2}$ – применяется формула для $x^{\deg(M)+1}$, умноженная на x и т.д. Например, вычисление $(x^3 + 2x + 3) \bmod (x + 2)$ с коэффициентами из $\mathbb{Z}_5$ состоит из

$x \bmod (x+2) \equiv x - (x+2) \equiv -2 \equiv 3 \pmod 5$;

$x^2 \bmod (x+2) \equiv 3x \equiv 3 \cdot 3 \equiv 9 \equiv 4 \pmod 5$;

$x^3 \bmod (x+2) \equiv 4x \equiv 4 \cdot 3 \equiv 12 \equiv 2 \pmod 5$;

$(x^3 + 2x + 3) \bmod (x+2) \equiv 2 + 2 \cdot 3 + 3 \equiv 11 \equiv 1 \pmod 5$.

Получение подматрицы матрицы A через левую, правую, верхнюю и нижнюю границу осуществляется путем задания четырех чисел i_1, $i_2 \geq i_1$, j_1, $j_2 \geq j_1$ возвращает матрицу B, элементы которой удовлетворяют формуле

$$B[i,j] = A[i_1 + i, j_1 + j], 1 \leq i \leq i_2 - i_1, 1 \leq j \leq j_2 - j_1. \quad (3.2)$$

В формуле (3.2) границы для индексов строк и столбцов даны в предположении, что строки и столбцы матрицы нумеруются с единицы.

Указание к вариантам:

1) Операции над конечными неупорядоченными множествами – объединение, пересечение, разность;

2) При реализации решета Эратосфена следует поставить в соответствие i – му биту число $2i + 3$ (биты нумеруются с нуля), причем 1 – если соответствующее биту число простое и 0 – если нет;

3) Вес Хемминга битового массива – количество единичных бит этого битового массива;

4) Умножение чисел через умножение многочленов осуществляется с использованием формулы из определения произведения многочленов, по следующей схеме на примере умножения чисел $132 \cdot 257$:

$(x^2 + 3x + 2) \cdot (2x^2 + 5x + 7) =$

$= 2x^4 + (5+6)x^3 + (7+15+4)x^2 + (21+10)x + 14 =$

$= 2x^4 + 11x^3 + 26x^2 + 31x + 14$;

$2x^4 + 11x^3 + 26x^2 + 31x + 14 \to 2x^4 + 11x^3 + 26x^2 + 32x + 4 \to$

$\to 2x^4 + 11x^3 + 29x^2 + 2x + 4 \to 2x^4 + 13x^3 + 9x^2 + 2x + 4 \to$

$\to 3x^4 + 3x^3 + 9x^2 + 2x + 4$;

$132 \cdot 257 = 33924$.

5) Варианты 5 – 9 были рассмотрены в лабораторной работе № 1;

6) Варианты 12-14 выполняются аналогично соответствующему алгоритму для чисел, причем в роли «нечетных» многочленов выступают многочлены с ненулевым свободным членом, а в роли «четных» - с нулевым.

7) Результатом алгоритма в варианте 15 является 64-х битное число, i – ый бит которого равен i – му коэффициенту линейной комбинации линейно зависимого набора строк (здесь строки нумеруются с нуля), например, матрице $\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$ соответствует число $\underbrace{0 \ldots 0}_{61\text{ бит}} 111_2 = 7_{10}$, поскольку сумма всех строк матрицы по модулю 2 равна нулевой строке.