МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КУБГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**Отчет**

**по практическому заданию №1**

**«Нахождение функции сложности алгоритмов»**

**по курсу**

**«ОЦЕНКА СЛОЖНОСТИ АЛГОРИТМОВ»**

Работу выполнила

Студентка 46 группы

Баева Д.Н.

Преподаватель:

Нигодин Е.А.

Краснодар 2024

**Цель работы:** найти функцию сложности алгоритмов:

1. Скалярное умножение векторов;
2. Перемножение матриц;
3. Сортировка массива (вставкой; выбором; пузырьковая).

Рассмотрим каждый из алгоритмов в отдельности и оценим их сложность.

**1. Скалярное умножение векторов.**

Скалярное произведение (умножение) векторов – это операция, результатом которой является число, полученное как сумма произведений соответствующих элементов векторов.

Пусть есть вектор и вектор , оба эти вектора имеют длину , тогда их скалярное произведение вычисляется по следующей формуле:

где - элементы вектора , а – элементы вектора .

Результирующий вектор также имеет длину .

Функция, вычисляющая скалярное произведение векторов представлена на рисунке 1.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1 – Функция вычисления скалярного произведения векторов.

Оценим сложность этого алгоритма.

**Асимптотическая оценка сложности**:

Для вычисления скалярного произведения двух векторов длины , мы выполняем умножений и сложений. Следовательно, общее количество операций составляет . Асимптотическая оценка сложности этого алгоритма — **O(n)** (линейная сложность).

**Количество операций**:

Генерация случайных векторов: 2 раза (по операций на каждый вектор). Вычисление скалярного произведения: операций (умножение и сложение). Итого: операций.

Таким образом, алгоритм имеет линейную сложность и выполняет примерно операций для вычисления скалярного произведения векторов.

**2. Перемножение матриц.**

Результатом операции умножения матриц Am×n и Bn×k будет матрица Cm×k такая, что элемент матрицы C, стоящий в i-той строке и j-том столбце (cij), равен сумме произведений элементов i-той строки матрицы A на соответствующие элементы -того столбца матрицы B**:** cij = ai1 · b1j + ai2 · b2j + ... + ain · bnj .

В нашем случае обе матрицы имеют размерность . Запишем определение в матричном виде.

Предположим, у нас есть две матрицы A и B размером :

Результирующая матрица C размеромбудет иметь следующий вид:

Функция, выполняющая перемножение матриц представлена на рисунке 2.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2 – Функция перемножения матриц.

Теперь выполним оценку сложности для данного алгоритма.

**Асимптотическая оценка сложности**:

Общая функция сложности этого алгоритма будет где n - количество строк в первой матрице, m - количество столбцов во второй матрице, p - количество столбцов в первой матрице (или количество строк во второй матрице). В данном случае, зная, что входные матрицы — квадратные матрицы размера , время работы составляет **O()** (то есть зависит кубически от размера матриц).

**Количество операций:**

Генерация случайных матриц: 2 раза (по операций на каждую матрицу). Вычисление произведения матриц:операций. Итого:операций.

**3. Сортировка массива (вставкой; выбором; пузырьковая).**

Рассмотрим и оценим каждую из сортировок отдельно.

**Сортировка вставкой** – элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов. Следовательно, аксиоматическая сложность составляет **O().** Количество операций (сравнений и обменов) примерно операций. [Это происходит из-за вложенных циклов, где внутренний цикл выполняется в среднем  раз для каждого элемента внешнего цикла](https://habr.com/ru/articles/173821/).

**Сортировка выбором** – на каждом шаге выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности. Аксиоматическая сложность также составляет **O().** Количество операций вновь примерно операций, так как здесь также есть вложенные циклы, которые перебирают элементы массива.

**Сортировка пузырьком** – пары элементов массива перебираются, и если 1-й элемент пары больше 2-го, элементы меняются местами. Этот процесс повторяется до тех пор, пока не обнаружится, что более не требуется выполнять перестановки (обмены). Аксиоматическая сложность: **O().** Количество операций примерно операций по аналогичным причинам.

Важно отметить, что данные алгоритмы являются учебными и на практике рекомендуется использовать более эффективные алгоритмы сортировки, такие как быстрая сортировка или сортировка слиянием.

Функции, представляющие собой каждую сортировку в отдельности, представлены на рисунках 3, 4 и 5 соответственно.

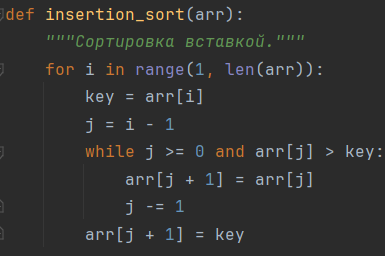


Рисунок 3 – Функция сортировки вставкой.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 4 – Функция сортировки выбором.

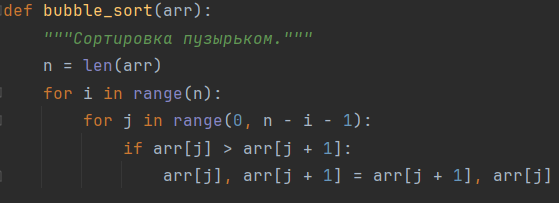


Рисунок 5 – Функция сортировки пузырьком.

**Результаты работы программы:**

На рисунках 6, 7, 8 представлен результат работы программ – скалярное перемножение векторов, перемножение матриц, сортировка массива (вставкой; выбором; пузырьковая).

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 6 – Результат скалярного перемножение векторов.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 7 – Результат перемножения матриц 3х3.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 8 – Результат сортировки массива разными способами.

**Листинг программ:**

Файл ScalarVectorMultiplication.py:

# скалярное умножение векторов  
  
import random  
  
def generate\_random\_vector(length):  
 *"""Генерирует случайный вектор заданной длины с целыми числами."""* return [random.randint(0, 100) for \_ in range(length)]  
  
  
def scalar\_product(vector1, vector2):  
 *"""Вычисляет скалярное произведение двух векторов."""* if len(vector1) != len(vector2):  
 raise ValueError("Векторы должны быть одинаковой длины")  
  
 # Вычисляем скалярное произведение вручную  
 result = 0  
 for i in range(len(vector1)):  
 result += vector1[i] \* vector2[i]  
  
 return result  
  
  
def main():  
 try:  
 length = int(input("Введите размерность векторов: "))  
 vector1 = generate\_random\_vector(length)  
 vector2 = generate\_random\_vector(length)  
  
 scalar\_result = scalar\_product(vector1, vector2)  
 print(f"Первый вектор: {vector1}")  
 print(f"Второй вектор: {vector2}")  
 print(f"Скалярное произведение: {scalar\_result:}")  
 except ValueError:  
 print("Пожалуйста, введите целое число для размерности векторов.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Файл MatrixMultiplication.py:

# перемножение матриц (n x n)  
  
import random  
  
def generate\_random\_matrix(n):  
 *"""Генерирует случайную матрицу размерности nxn с целочисленными значениями."""* return [[random.randint(0, 100) for \_ in range(n)] for \_ in range(n)]  
  
def matrix\_multiply(A, B):  
 *"""Вычисляет произведение двух матриц."""* n = len(A)  
 p = len(B[0])  
 m = len(B)  
  
 # Создаем матрицу C размерности nxp, заполняем нулями  
 C = [[0] \* p for \_ in range(n)]  
  
 for i in range(n):  
 for j in range(p):  
 # Вычисляем сумму произведений элементов A[i][k] и B[k][j]  
 for k in range(m):  
 C[i][j] += A[i][k] \* B[k][j]  
  
 return C  
  
def main():  
 try:  
 n = int(input("Введите размерность матриц (n): "))  
 matrix1 = generate\_random\_matrix(n)  
 matrix2 = generate\_random\_matrix(n)  
  
 result\_matrix = matrix\_multiply(matrix1, matrix2)  
 print("Первая матрица:")  
 for row in matrix1:  
 print(row)  
 print("\nВторая матрица:")  
 for row in matrix2:  
 print(row)  
 print("\nРезультат умножения матриц:")  
 for row in result\_matrix:  
 print(row)  
 except ValueError:  
 print("Пожалуйста, введите целое число для размерности матриц.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Файл ArraySort.py:

# сортировка массива (вставкой; выбором; пузырьковая)  
  
import random  
  
def generate\_random\_array(n):  
 *"""Генерирует случайный массив целых чисел заданной длины."""* return [random.randint(0, 100) for \_ in range(n)]  
  
def insertion\_sort(arr):  
 *"""Сортировка вставкой."""* for i in range(1, len(arr)):  
 key = arr[i]  
 j = i - 1  
 while j >= 0 and arr[j] > key:  
 arr[j + 1] = arr[j]  
 j -= 1  
 arr[j + 1] = key  
  
def selection\_sort(arr):  
 *"""Сортировка выбором."""* for i in range(len(arr)):  
 min\_index = i  
 for j in range(i + 1, len(arr)):  
 if arr[j] < arr[min\_index]:  
 min\_index = j  
 arr[i], arr[min\_index] = arr[min\_index], arr[i]  
  
def bubble\_sort(arr):  
 *"""Сортировка пузырьком."""* n = len(arr)  
 for i in range(n):  
 for j in range(0, n - i - 1):  
 if arr[j] > arr[j + 1]:  
 arr[j], arr[j + 1] = arr[j + 1], arr[j]  
  
def main():  
 try:  
 n = int(input("Введите размерность массива: "))  
 my\_array = generate\_random\_array(n)  
  
 print("Исходный массив:")  
 print(my\_array)  
  
 # Копируем массив для каждой сортировки  
 array\_insertion = my\_array.copy()  
 array\_selection = my\_array.copy()  
 array\_bubble = my\_array.copy()  
  
 insertion\_sort(array\_insertion)  
 selection\_sort(array\_selection)  
 bubble\_sort(array\_bubble)  
  
 print("\nОтсортированный массив (вставкой):")  
 print(array\_insertion)  
 print("\nОтсортированный массив (выбором):")  
 print(array\_selection)  
 print("\nОтсортированный массив (пузырьком):")  
 print(array\_bubble)  
 except ValueError:  
 print("Пожалуйста, введите целое число для размерности массива.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()