ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 5

«Сортировки»

Выполнил работу

Мельников Олег

Академическая группа №J3112

Санкт-Петербург

2024

1. Введение

В данной работе рассматривается реализация и анализ алгоритмов сортировки, которые используются в программировании для упорядочивания данных. Целью работы является понимание различных алгоритмов сортировки, их эффективности и применения в зависимости от конкретных условий задачи. Для достижения этой цели будут реализованы три алгоритма: пузырьковая сортировка, сортировка слиянием и сортировка подсчетом.

Задачи, которые предстоит решить:

Реализовать алгоритм пузырьковой сортировки и проанализировать его временную и пространственную сложность.

Реализовать алгоритм сортировки слиянием и оценить его эффективность.

Реализовать алгоритм сортировки подсчетом, который подходит для сортировки чисел в ограниченном диапазоне, и сравнить его производительность.

1. Теоретическая подготовка

Сортировка

Сортировка является ключевым процессом в обработке данных, который упрощает поиск, анализ и визуализацию информации. Существует множество алгоритмов сортировки, каждый из которых имеет свои преимущества и недостатки. Основные критерии выбора алгоритма включают временную и пространственную сложность.

Типы данных

В данной работе используются одномерные массивы (векторы) для хранения целых чисел. В C++ вектор является динамическим массивом, который позволяет изменять размер в процессе выполнения программы.

Алгоритмы

1. **Пузырьковая сортировка**:
   * Простая и интуитивно понятная сортировка, которая работает по принципу многократного прохода по массиву, последовательно сравнивая и меняя местами соседние элементы.
   * Временная сложность: O(N^2) в худшем и среднем случае; O(N) в лучшем (если массив уже отсортирован).
   * Пространственная сложность: O(1).
2. **Сортировка слиянием**:
   * Разделяющий и завоевывающий алгоритм, который рекурсивно делит массив пополам, сортирует каждую половину, а затем объединяет их обратно.
   * Временная сложность: O(N log N) в худшем, среднем и лучшем случаях, что делает этот алгоритм более эффективным для больших наборов данных.
   * Пространственная сложность: O(N), так как необходим дополнительный массив для хранения отсортированных данных на каждом этапе слияния.
3. **Сортировка подсчетом**:
   * Эффективный алгоритм, подходящий только для отсортировки целых чисел в ограниченном диапазоне (k) по сравнению с размером массива (N).
   * Временная сложность: (O(n + k)).
   * Пространственная сложность: O(n + k), так как требуется дополнительная память для хранения счетчиков.

Эти алгоритмы служат примером различных подходов к решению задачи сортировки, и каждый из них применим в разных сценариях в зависимости от требований к скорости и объему памяти. В последующих разделах будут приведены конкретные реализации этих алгоритмов на языке программирования C++.

1. Реализация

Этап 1: Подготовка среды разработки

Для реализации алгоритмов сортировки использовался язык программирования C++. В качестве среды разработки была выбрана Visual Studio Code.

Этап 2: Исполнение алгоритмов

Для каждой задачи был реализован соответствующий алгоритм сортировки в отдельном модуле, что позволяет изолированно тестировать и изменять код каждого алгоритма.

2.1 Реализация пузырьковой сортировки

Особенности реализации:

* Используется механизм обмена элементов через swap, что делает код более чистым и читабельным.
* Введена логика проверки, были сделаны обмены, для оптимизации. Если массив уже отсортирован, выполнение прекращается.

2.2 Реализация сортировки слиянием

Исходный код сортировки слиянием был разделён на две основные функции: рекурсивную функцию сортировки и функцию слияния:

Особенности реализации:

* Введены вспомогательные векторы для хранения временных данных перед объединением, что способствует ясности и упрощает слияние.
* Рекурсивный подход используется для деления массива на части вплоть до единичных элементов.

2.3 Реализация сортировки подсчетом

Алгоритм сортировки подсчетом требует знания диапазона значений, который задается максимальным элементом (k) в массиве. Это позволяет создать вспомогательный массив count, размером (k + 1), что делает алгоритм эффективным для обработки данных с известным диапазоном.

Этап 3: Тестирование алгоритмов

После реализации каждого алгоритма была проведена его проверка с использованием различных наборов данных. Оценивалась корректность работы, а также время выполнения для различных размеров массивов. Использовались стандартные инструменты C++ для вывода в консоль, что позволяло легко отслеживать результаты.

1. Экспериментальная часть

Для тестирования были использованы три алгоритма сортировки: пузырьковая сортировка, сортировка слиянием и сортировка подсчетом. Измерения времени проводились для различных размерностей входных данных, начиная от 1 элемента и заканчивая 100 элементами, с интервалом в 10 элементов.

**Таблица 1 - Подсчет времени выполнения реализованных алгоритмов**

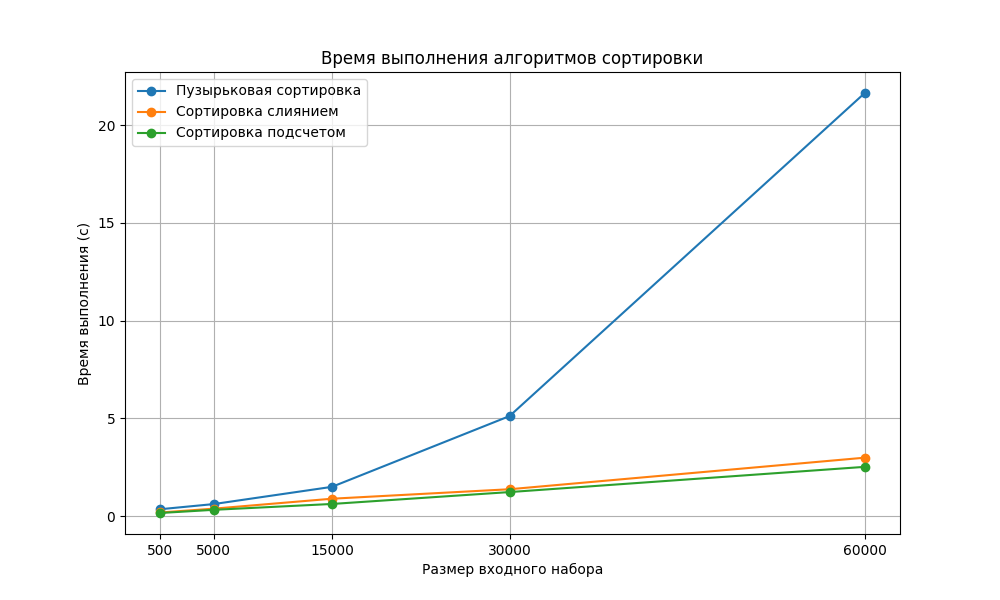
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размер входных данных | 500 | 5000 | 15000 | 30000 | 60000 |
| Пузырьковая сортировка | 0.357 | 0.619 | 1.501 | 5.12 | 21.641 |
| Сортировка слиянием | 0.190 | 0.385 | 0.894 | 1.379 | 2.993 |
| Сортировка подсчетом | 0.164 | 0.326 | 0.626 | 1.234 | 2.524 |

Анализ

* **Пузырьковая сортировка**:
  + Временная сложность: O(N^2), что подтверждается результатами — время значительно увеличивается с ростом входного набора.
  + Пространственная сложность: O(1), так как для сортировки используется только несколько переменных.
* **Сортировка слиянием**:
  + Временная сложность: O(N log N), что видно из линейного роста времени выполнения при увеличении размера массива.
  + Пространственная сложность: O(N), требуется дополнительный массив для временного хранения данных.
* **Сортировка подсчетом**:
  + Временная сложность: O(N + k), о чем свидетельствует стабильное время выполнения для массивов с небольшим диапазоном значений.
  + Пространственная сложность: O(N + k), используется массив для хранения частот и выходной массив.

**График зависимости времени от числа элементов**

График, представляющий время выполнения алгоритмов в зависимости от размеров входных данных, представлен ниже.



Изображение 1 - График времени выполнения алгоритмов сортировки

Анализ графика и таблицы

Анализ результатов показывает, что:

1. **Пузырьковая сортировка** демонстрирует экспоненциальный рост времени выполнения по мере увеличения количества элементов. Это соответствует теоретической оценке O(N^2) и делает данный алгоритм неэффективным для больших наборов данных.
2. **Сортировка слиянием** значительно более эффективна, время выполнения сохраняется близким к O(N log N), что подтверждает её производительность, особенно для больших массивов.
3. **Сортировка подсчетом** показывает наименьшее время выполнения, особенно для наборов данных с ограниченным диапазоном значений. Это подтверждает ее эффективность в ситуациях, где диапазон величин значительно меньше общего количества элементов.

Таким образом, графическое представление и табличные данные подтверждают теоретические оценки производительности алгоритмов, демонстрируя, что выбор метода сортировки имеет критическое значение в зависимости от конкретных условий и ограничений задачи.

1. Заключение

В ходе выполнения данной работы были реализованы и протестированы три различных алгоритма сортировки: пузырьковая сортировка, сортировка слиянием и сортировка подсчетом. Цель работы заключалась в понимании реализуемых алгоритмов, их сравнении с точки зрения временной и пространственной сложности, а также в практике их применения на реальных данных.

Тестирование алгоритмов на различных наборах данных показало значительные различия в их производительности. Пузырьковая сортировка, хоть и является простым для реализации методом, продемонстрировала наименьшую эффективность, особенно при увеличении размера массива, что подтверждает теоретическую оценку сложности O(N^2). В то же время сортировка слиянием показала стабильные результаты с временной сложностью O(N log N), что делает её предпочтительным выбором для сортировки больших массивов. Сортировка подсчетом продемонстрировала наилучшие результаты в условиях ограниченного диапазона значений, подтверждая свою эффективность с временной сложностью O(N + k).

Полученные результаты совпадают с ожидаемыми теоретическими оценками, что подтверждает правильность реализованных алгоритмов и их соответствие установленным требованиям.

Таким образом, работа не только достигла своей основной цели, но и открыла новые перспективы для углубленного изучения и оптимизации алгоритмов сортировки.

1. Приложения

ПРИЛОЖЕНИЕ А

Листинг кода файла 1.cpp

#include <iostream>

#include <vector>

#include <fstream>

#include <sstream>

#include <ctime>

using namespace std;

void bubbleSort(vector<int> &arr) {

    int n = arr.size();

    bool swapped;

    for (int i = 0; i < n - 1; i++) {

        swapped = false;

        for (int j = 0; j < n - i - 1; j++) {

            if (arr[j] > arr[j + 1]) {

                swap(arr[j], arr[j + 1]); // Обмен элементов

                swapped = true;

            }

        }

        // Если на этой итерации не было обменов, массив уже отсортирован

        if (!swapped) break;

    }

}

int main() {

    vector<int> arr;

    ifstream input;

    input.open("C:\\Users\\oleg-\\lr52345.txt");

    if (!input) {

        cout << "Не";

        return 1;

    }

    string line;

    while (getline(input, line)) {

        stringstream ss(line);

        string number;

        while (getline(ss, number, '\t')) {

            arr.push\_back(stoi(number));

        }

    }

    bubbleSort(arr);

    cout << "Отсортированный массив: ";

    for (int el : arr) {

        cout << el << " ";

    }

    cout << "runtime = " << clock()/1000.0 << endl; // время работы программы

    return 0;

}

Листинг кода файла 2.cpp

#include <iostream>

#include <vector>

#include <fstream>

#include <sstream>

#include <ctime>

using namespace std;

void merge(vector<int> &arr, int left, int mid, int right) {

    int n1 = mid - left + 1;

    int n2 = right - mid;

    vector<int> L(n1), R(n2);

    for (int i = 0; i < n1; i++)

        L[i] = arr[left + i];

    for (int j = 0; j < n2; j++)

        R[j] = arr[mid + 1 + j];

    int i = 0, j = 0, k = left;

    while (i < n1 && j < n2) {

        if (L[i] <= R[j]) {

            arr[k] = L[i];

            i++;

        } else {

            arr[k] = R[j];

            j++;

        }

        k++;

    }

    while (i < n1) {

        arr[k] = L[i];

        i++;

        k++;

    }

    while (j < n2) {

        arr[k] = R[j];

        j++;

        k++;

    }

}

void mergeSort(vector<int> &arr, int left, int right) {

    if (left < right) {

        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);

        mergeSort(arr, mid + 1, right);

        merge(arr, left, mid, right);

    }

}

int main() {

    vector<int> arr;

    ifstream input;

    input.open("C:\\Users\\oleg-\\lr52345.txt");

    if (!input) {

        cout << "Не";

        return 1;

    }

    string line;

    while (getline(input, line)) {

        stringstream ss(line);

        string number;

        while (getline(ss, number, '\t')) {

            arr.push\_back(stoi(number));

        }

    }

    int n = arr.size();

    mergeSort(arr, 0, n - 1);

    cout << "Отсортированный массив: ";

    for (int el : arr) {

        cout << el << " ";

    }

    cout << "runtime = " << clock()/1000.0 << endl; // время работы программы

    return 0;

}

Листинг кода файла 3.cpp

#include <iostream>

#include <vector>

#include <fstream>

#include <sstream>

#include <algorithm>

#include <ctime>

using namespace std;

void countingSort(vector<int> &arr, int k) {

    // Шаг 1: Создаём вектор count для хранения количества вхождений

    vector<int> count(k + 1, 0);

    // Шаг 2: Считаем количество вхождений каждого числа

    for (int i = 0; i < arr.size(); i++) {

        count[arr[i]]++;

    }

    // Шаг 3: Заполняем выходной массив с учетом количества вхождений

    vector<int> output;

    for (int i = 0; i <= k; i++) {

        for (int j = 0; j < count[i]; j++) {

            output.push\_back(i);  // Добавляем i в выходной массив count[i] раз

        }

    }

    // Шаг 4: Обновляем оригинальный массив arr

    arr = output;

}

int main() {

    vector<int> arr;

    ifstream input("C:\\Users\\oleg-\\lr52345.txt");

    if (!input) {

        cout << "Не удалось открыть файл." << endl;

        return 1;

    }

    string line;

    while (getline(input, line)) {

        stringstream ss(line);

        string number;

        while (getline(ss, number, '\t')) {

            arr.push\_back(stoi(number));

        }

    }

    int k = \*max\_element(arr.begin(), arr.end()); // Максимальное значение в массиве

    countingSort(arr, k);

    cout << "Отсортированный массив: ";

    for (int el : arr) {

        cout << el << " ";

    }

    cout << endl;

    cout << "runtime = " << clock()/1000.0 << endl; // время работы программы

    return 0;

}