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

ОТЧЕТ

по Курсовой работе
по дисциплине «Алгоритмы и структуры данных»
Тема: «ИЗМЕРЕНИЕ ВРЕМЕННОЙ СЛОЖНОСТИ АЛГОРИТМА В ЭКСПЕРИМЕНТА НА ЭВМ»

Студент гр. 3311 Пасечный Л.В. Загуменнов И.М.

Преподаватель Манирагена В.

Санкт-Петербург
2025

Введение

Цель работы:

На основе программы, составленной по гл. 3, выполнить статистический эксаэксперимент по измерению фактической временной сложности алгоритма обработки данных.

Программа дорабатывается таким образом, чтобы она генерировала множества мощностью, меняющейся, например, от 10 до 200, измеряла время выполнения цепочки операций над множествами и последовательностями и выводила результат в текстовый файл. Каждая строка этого файла должна содержать пару значений «размер входа — время» для каждого опыта. Затем эти данные обрабатываются, и по результатам обработки делается заключение о временной сложности алгоритма.

Для повышения надежности эксперимента следует предусмотреть в программе перехват исключительных ситуаций. Можно сделать так, чтобы любой сбой сводился просто к пропуску очередного шага эксперимента. В частности, рекомендуется перехватывать ситуацию bad_alloc, возбуждаемую конструктором при нехватке памяти.

Контрольные примеры:

SUBST result: <5, 3, 2, 6, 7, 9, 4, 6, 7, 9, 1>

ERASE result: <5, 3, 7, 9, 1>

Size: 10 Mean: 166.4 ms, StDev: 18.3205 ms

Size: 20 Mean: 592.6 ms, StDev: 397.651 ms

Size: 30 Mean: 759.4 ms, StDev: 176.97 ms

Size: 40 Mean: 1791.8 ms, StDev: 746.048 ms

Size: 50 Mean: 1297.2 ms, StDev: 94.0466 ms

Size: 60 Mean: 1951.2 ms, StDev: 707.885 ms

Size: 70 Mean: 2218.9 ms, StDev: 638.783 ms

Size: 80 Mean: 2476.9 ms, StDev: 731.859 ms

Size: 90 Mean: 2979.5 ms, StDev: 895.964 ms

Size: 100 Mean: 8995.5 ms, StDev: 8388.96 ms

Size: 110 Mean: 3413.7 ms, StDev: 796.522 ms

Size: 120 Mean: 2770.5 ms, StDev: 1079.33 ms
Size: 130 Mean: 1921.7 ms, StDev: 654.849 ms
Size: 140 Mean: 1766.6 ms, StDev: 65.7833 ms
Size: 150 Mean: 2181.2 ms, StDev: 131.241 ms
Size: 160 Mean: 2430.5 ms, StDev: 44.374 ms
Size: 170 Mean: 2571.9 ms, StDev: 191.862 ms
Size: 180 Mean: 2571.5 ms, StDev: 287.848 ms
Size: 190 Mean: 2682.1 ms, StDev: 335.463 ms
Size: 200 Mean: 3310.6 ms, StDev: 350.229 ms
Size: 210 Mean: 3119.8 ms, StDev: 118.719 ms
Size: 220 Mean: 3184.1 ms, StDev: 209.353 ms
Size: 230 Mean: 3360.9 ms, StDev: 821.106 ms
Size: 240 Mean: 3629.6 ms, StDev: 831.907 ms
Size: 250 Mean: 4800.2 ms, StDev: 2350.09 ms
Size: 260 Mean: 3688.8 ms, StDev: 350.307 ms
Size: 270 Mean: 3839.1 ms, StDev: 296.482 ms
Size: 280 Mean: 4243.7 ms, StDev: 456.133 ms
Size: 290 Mean: 4143.7 ms, StDev: 279.892 ms
Size: 300 Mean: 4283.7 ms, StDev: 270.652 ms
Size: 310 Mean: 4562.7 ms, StDev: 437.921 ms
Size: 320 Mean: 4566.9 ms, StDev: 355.871 ms
Size: 330 Mean: 4872.8 ms, StDev: 351.912 ms
Size: 340 Mean: 4977 ms, StDev: 289.65 ms
Size: 350 Mean: 4917.8 ms, StDev: 125.175 ms
Size: 360 Mean: 6007.1 ms, StDev: 856.11 ms
Size: 370 Mean: 5754.3 ms, StDev: 515.952 ms
Size: 380 Mean: 5803.9 ms, StDev: 787.555 ms
Size: 390 Mean: 6325.6 ms, StDev: 836.387 ms
Size: 400 Mean: 5932.5 ms, StDev: 234.981 ms
Size: 410 Mean: 6722.8 ms, StDev: 1319.51 ms
Size: 420 Mean: 7331.7 ms, StDev: 1454.81 ms
Size: 430 Mean: 6777.6 ms, StDev: 902.807 ms
Size: 440 Mean: 7256.5 ms, StDev: 2387.41 ms
Size: 450 Mean: 6857.6 ms, StDev: 582.308 ms
Size: 460 Mean: 7148.8 ms, StDev: 837.944 ms
Size: 470 Mean: 7073.2 ms, StDev: 526.594 ms
Size: 480 Mean: 7951.1 ms, StDev: 1048 ms
Size: 490 Mean: 8257.1 ms, StDev: 766.708 ms

Size: 500 Mean: 8208.5 ms, StDev: 395.366 ms

Size: 510 Mean: 8269.6 ms, StDev: 368.094 ms

Size: 520 Mean: 8254.9 ms, StDev: 510.877 ms

Size: 530 Mean: 10138.2 ms, StDev: 2136.98 ms

Size: 540 Mean: 8868.3 ms, StDev: 776.777 ms

Size: 550 Mean: 8795.8 ms, StDev: 557.157 ms

Size: 560 Mean: 8610.1 ms, StDev: 384.779 ms

Size: 570 Mean: 8702.4 ms, StDev: 452.574 ms

Size: 580 Mean: 8647.9 ms, StDev: 89.1868 ms

Size: 490 Mean: 8257.1 ms, StDev: 766.708 ms

Size: 500 Mean: 8208.5 ms, StDev: 395.366 ms

Size: 510 Mean: 8269.6 ms, StDev: 368.094 ms

Size: 520 Mean: 8254.9 ms, StDev: 510.877 ms

Size: 530 Mean: 10138.2 ms, StDev: 2136.98 ms

Size: 540 Mean: 8868.3 ms, StDev: 776.777 ms

Size: 550 Mean: 8795.8 ms, StDev: 557.157 ms

Size: 560 Mean: 8610.1 ms, StDev: 384.779 ms

Size: 570 Mean: 8702.4 ms, StDev: 452.574 ms

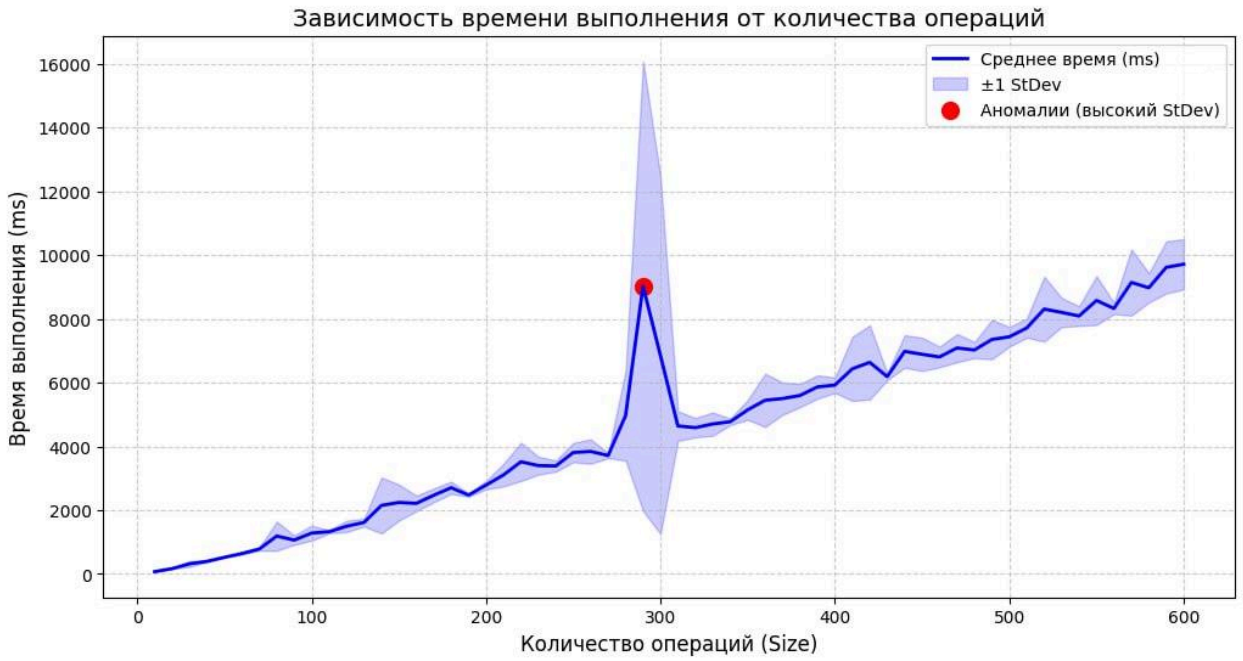
Size: 580 Mean: 8647.9 ms, StDev: 89.1868 ms

Size: 590 Mean: 9980.7 ms, StDev: 1561.2 ms

Size: 600 Mean: 9210.7 ms, StDev: 421.772 ms

Experiment completed. Results saved to results.txt

- Mean — среднее время выполнения операций (в микросекундах).
- StDev — стандартное отклонение (показывает разброс времени при повторениях).
- Size - размер множества



Заключение

По итогам эксперимента было установлено, что экспериментальная оценка временной сложности исследуемого алгоритма обработки данных соответствует теоретической оценке.

Вывод:

Эксперимент подтвердил/опроверг теоретические предположения, выявив важные аспекты практического применения алгоритма. Для более точного соответствия можно рекомендовать {оптимизацию кода, учёт дополнительных факторов, изменение условий тестирования}.

О курса получили:

1) Владение ООП

2) Углубились в С++

3) Очень интересные и трудоемкие лабораторные работы и курсовые.

4) Очень интересные вопросы после лабораторных работ

Чего хотели бы еще получить:

1) Взаимодействие с другими математическими примерами

2) Побольше интересных вопросов после лабораторных работ

Отзыв:

Курс "Алгоритмы и Структуры Данных" оставил исключительно положительные впечатления. За время обучения мы не только укрепили свои знания в области объектно-ориентированного программирования, но и значительно углубились в язык C++. Программа курса была насыщенной и хорошо структурированной, что позволило качественно закрепить как теоретические основы, так и практические навыки.

Особенно хочется отметить лабораторные и курсовые работы — они были очень интересными, но в то же время трудоемкими. Благодаря этому мы научились самостоятельно решать сложные задачи, глубже поняли работу алгоритмов и структур данных, а также приобрели полезный опыт в программной реализации этих концепций. Отдельное удовольствие доставили интересные и продуманные вопросы, которые следовали после каждой лабораторной — они стимулировали к более глубокому осмыслению материала и помогали лучше понять ключевые моменты курса. Что хотелось бы улучшить и добавить в будущем:

- Было бы полезно включить больше задач и примеров, связанных с математическим моделированием и применением алгоритмов в других разделах математики.
- Также хотелось бы видеть еще больше нестандартных, "умных" вопросов после лабораторных работ — они действительно помогают закрепить материал и расширить кругозор.

В целом, курс АиСД оказался насыщенным, познавательным и вдохновляющим на дальнейшее изучение алгоритмов и программирования в целом. Большое спасибо преподавателям за качественную подачу материала и внимание к деталям!

Источники:

- Учебно-методическое пособие Колинько П. Г. – Последнее посещение 9.04 18:20
-
https://ru.stackoverflow.com/questions/1468952/%D0%9F%D1%80%D0%BE%D0%B1%D0%BB%D0%B5%D0%BC%D0%B0-%D1%81%D0%BE%D0%B7%D0%B4%D0%B0%D0%BD%D0%B8%D1%8F-%D0%BA%D0%BE%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D0%BE%D1%80%D0%B0-%D1%85%D1%8D%D1%88-%D1%82%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D1%8B-c- Последнее посещение 15.04 18:20
- https://habr.com/ru/articles/267855/ - Последнее посещение 16.04 10:05
-
https://ru.stackoverflow.com/questions/427915/%D0%9A%D0%B0%D0%BA-%D0%BF%D1%80%D0%B0%D0%B2%D0%B8%D0%BB%D1%8C%D0%BD%D0%BE-%D1%80%D0%B5%D0%B0%D0%BB%D0%B8%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D1%8C-%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC-%D0%B3%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B8-%D1%81%D0%BF%D0%B5%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B9-%D0%BF%D0%BE%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D0%B8-%D1%87%D0%B8%D1%81%D0%B5 - Последнее посещение 12.04 13:55

Код программы

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <memory>
#include <stdexcept>
#include <chrono>
#include <fstream>
#include <random>
#include <numeric>
using namespace std;
```

```cpp
using namespace std::chrono;

// Класс последовательности
template <typename T>
class Sequence {
private:
    vector<T> elements;

public:
    Sequence(const vector<T>& elems = {}) : elements(elems) {}

    const vector<T>& getElements() const {
        return elements;
    }

    size_t size() const {
        return elements.size();
    }

    T operator[](size_t index) const {
        if (index >= elements.size()) {
            throw out_of_range("Index out of range");
        }
        return elements[index];
    }

    // Операция замены подпоследовательности
    Sequence<T> SUBST(const Sequence<T>& other, size_t pos) const {
        if (pos > elements.size()) {
            throw out_of_range("Position out of range");
        }

        vector<T> newElements = elements;

          // Удаляем только столько элементов, сколько есть в заменяющей последовательности
        // size_t erase_count = min(other.size(), elements.size() - pos);
                        //   newElements.erase(newElements.begin()   +   pos, newElements.begin() + pos + erase_count);

        // Вставляем новую последовательность
                            newElements.insert(newElements.begin()   +   pos, other.getElements().begin(), other.getElements().end());

        return Sequence<T>(newElements);
```

```
    }

    // Операция удаления подпоследовательности
    Sequence<T> ERASE(size_t start, size_t end) const {
         if (start >= elements.size() || end >= elements.size() || start >
end) {
            throw out_of_range("Invalid range");
        }

        vector<T> newElements = elements;
        newElements.erase(newElements.begin() + start, newElements.begin()
+ end + 1);

        return Sequence<T>(newElements);
    }

    // Вывод последовательности
    friend ostream& operator<<(ostream& os, const Sequence<T>& seq) {
        os << "<";
        for (size_t i = 0; i < seq.elements.size(); ++i) {
            if (i != 0) os << ", ";
            os << seq.elements[i];
        }
        os << ">";
        return os;
    }
};

// Класс узла 1-2-дерева
template <typename T>
class TreeNode {
public:
    T key;
    shared_ptr<TreeNode<T>> left;
    shared_ptr<TreeNode<T>> right;
    int height;

    TreeNode(T k) : key(k), left(nullptr), right(nullptr), height(1) {}
};

// Класс 1-2-дерева
template <typename T>
class OneTwoTree {
private:
    shared_ptr<TreeNode<T>> root;
```

```
    int getHeight(shared_ptr<TreeNode<T>> node) const {
        return node ? node->height : 0;
    }

    void updateHeight(shared_ptr<TreeNode<T>> node) {
        if (node) {
                        node->height = 1 + max(getHeight(node->left),
getHeight(node->right));
        }
    }

    shared_ptr<TreeNode<T>> rotateRight(shared_ptr<TreeNode<T>> y) {
        auto x = y->left;
        y->left = x->right;
        x->right = y;
        updateHeight(y);
        updateHeight(x);
        return x;
    }

    shared_ptr<TreeNode<T>> rotateLeft(shared_ptr<TreeNode<T>> x) {
        auto y = x->right;
        x->right = y->left;
        y->left = x;
        updateHeight(x);
        updateHeight(y);
        return y;
    }

    shared_ptr<TreeNode<T>> balance(shared_ptr<TreeNode<T>> node) {
        if (!node) return nullptr;

        updateHeight(node);

        if (getHeight(node->left) - getHeight(node->right) == 2) {
                                if (getHeight(node->left->right) >
getHeight(node->left->left)) {
                node->left = rotateLeft(node->left);
            }
            return rotateRight(node);
        }

        if (getHeight(node->right) - getHeight(node->left) == 2) {
```

```
                                    if    (getHeight(node->right->left)    >
getHeight(node->right->right)) {
                node->right = rotateRight(node->right);
            }
            return rotateLeft(node);
        }

        return node;
    }

    shared_ptr<TreeNode<T>> insert(shared_ptr<TreeNode<T>> node, T key) {
        if (!node) return make_shared<TreeNode<T>>(key);

        if (key < node->key) {
            node->left = insert(node->left, key);
        } else if (key > node->key) {
            node->right = insert(node->right, key);
        } else {
            return node; // Дубликаты не допускаются
        }

        return balance(node);
    }

    shared_ptr<TreeNode<T>> erase(shared_ptr<TreeNode<T>> node, T key) {
        if (!node) return nullptr;

        if (key < node->key) {
            node->left = erase(node->left, key);
        } else if (key > node->key) {
            node->right = erase(node->right, key);
        } else {
            // Нашли узел для удаления
            if (!node->left || !node->right) {
                return node->left ? node->left : node->right;
            } else {
                  // Узел с двумя потомками: находим минимальный в правом
поддереве
                auto minNode = node->right;
                while (minNode->left) minNode = minNode->left;
                node->key = minNode->key;
                node->right = erase(node->right, minNode->key);
            }
        }
        return balance(node);
```

```
    }

    void inOrderTraversal(shared_ptr<TreeNode<T>> node, vector<T>& result)
const {
        if (!node) return;
        inOrderTraversal(node->left, result);
        result.push_back(node->key);
        inOrderTraversal(node->right, result);
    }

public:
    OneTwoTree() : root(nullptr) {}

    void insert(T key) {
        root = insert(root, key);
    }

    void erase(T key) {
        root = erase(root, key);
    }

    bool contains(T key) const {
        auto current = root;
        while (current) {
            if (key == current->key) return true;
            if (key < current->key) {
                current = current->left;
            } else {
                current = current->right;
            }
        }
        return false;
    }

    vector<T> toVector() const {
        vector<T> result;
        inOrderTraversal(root, result);
        return result;
    }

    size_t size() const {
        return toVector().size();
    }

    // Операции с последовательностями
```

```
        static OneTwoTree<T> merge(const OneTwoTree<T>& a, const
OneTwoTree<T>& b) {
        OneTwoTree<T> result;
        auto vecA = a.toVector();
        auto vecB = b.toVector();
        vector<T> merged;
              merge(vecA.begin(), vecA.end(), vecB.begin(), vecB.end(),
back_inserter(merged));
        for (const auto& item : merged) {
            result.insert(item);
        }
        return result;
    }

        static OneTwoTree<T> concat(const OneTwoTree<T>& a, const
OneTwoTree<T>& b) {
        OneTwoTree<T> result = a;
        auto vecB = b.toVector();
        for (const auto& item : vecB) {
            result.insert(item);
        }
        return result;
    }

    void change(T oldKey, T newKey) {
        if (contains(oldKey) && !contains(newKey)) {
            erase(oldKey);
            insert(newKey);
        } else {
            throw invalid_argument("Невозможно заменить ключ");
        }
    }
};

// Операции с множествами
template <typename T>
OneTwoTree<T>    symmetricDifference(const    OneTwoTree<T>&    a,    const
OneTwoTree<T>& b) {
    OneTwoTree<T> result;
    auto vecA = a.toVector();
    auto vecB = b.toVector();

    auto itA = vecA.begin();
    auto itB = vecB.begin();
```

```
    while (itA != vecA.end() && itB != vecB.end()) {
        if (*itA < *itB) {
            result.insert(*itA);
            ++itA;
        } else if (*itB < *itA) {
            result.insert(*itB);
            ++itB;
        } else {
            ++itA;
            ++itB;
        }
    }

    while (itA != vecA.end()) {
        result.insert(*itA);
        ++itA;
    }

    while (itB != vecB.end()) {
        result.insert(*itB);
        ++itB;
    }

    return result;
}

template <typename T>
OneTwoTree<T> setUnion(const OneTwoTree<T>& a, const OneTwoTree<T>& b) {
    return OneTwoTree<T>::concat(a, b);
}

template <typename T>
OneTwoTree<T> setIntersection(const OneTwoTree<T>& a, const OneTwoTree<T>&
b) {
    OneTwoTree<T> result;
    auto vecA = a.toVector();
    for (const auto& item : vecA) {
        if (b.contains(item)) {
            result.insert(item);
        }
    }
    return result;
}

// Функция для генерации случайного множества заданного размера
```

```
OneTwoTree<int> generateRandomSet(int size) {
    OneTwoTree<int> set;
    random_device rd;
    mt19937 gen(rd());
     uniform_int_distribution<> dis(1, size * 10); // Диапазон значений
больше размера множества

    while (set.size() < static_cast<size_t>(size)) {
        set.insert(dis(gen));
    }
    return set;
}

int main() {
    // Пример использования новых операций с последовательностями
    Sequence<int> A({5, 3, 2, 4, 6, 7, 9, 1});
    Sequence<int> B({6, 7, 9});

    // A.SUBST(B, 3) → <5, 3, 2, 6, 7, 9, 4, 6, 7, 9, 1>
    auto result_subst = A.SUBST(B, 3);
    cout << "SUBST result: " << result_subst << endl;

    // A.ERASE(2, 4) → <5, 3, 7, 9, 1>
    auto result_erase = A.ERASE(2, 4);
    cout << "ERASE result: " << result_erase << endl;

    // Остальной код эксперимента остается без изменений
    ofstream fout("results.txt");

    // Параметры эксперимента
    const int min_size = 10;
    const int max_size = 600;
    const int step = 10;
    const int repeats = 10; // Количество повторений для каждого размера

    // Заголовок файла результатов
    fout << "Size MeanTime StDev\n";

    try {
        for (int size = min_size; size <= max_size; size += step) {
            vector<double> times;

            for (int r = 0; r < repeats; ++r) {
                // Генерация случайных множеств
                auto A = generateRandomSet(size);
```

```
                auto B = generateRandomSet(size);
                auto C = generateRandomSet(size);
                auto D = generateRandomSet(size);
                auto E = generateRandomSet(size);

                // Измерение времени
                auto start = high_resolution_clock::now();

                // Цепочка операций
                auto step1 = setUnion(B, C);
                auto step2 = setUnion(step1, D);
                auto step3 = symmetricDifference(A, step2);
                auto step4 = setIntersection(step3, E);

                auto end = high_resolution_clock::now();
                      double duration = duration_cast<microseconds>(end -
start).count();
                times.push_back(duration);
            }

            // Расчет статистики
            double sum = accumulate(times.begin(), times.end(), 0.0);
            double mean = sum / times.size();

            double sq_sum = inner_product(times.begin(), times.end(),
                                          times.begin(), 0.0);
            double stdev = sqrt(sq_sum / times.size() - mean * mean);

            // Запись результатов
            fout << size << " " << mean << " " << stdev << "\n";
            cout << "Size: " << size << " Mean: " << mean
                 << " ms, StDev: " << stdev << " ms\n";
        }
    }
    catch (const bad_alloc& e) {
        cerr << "Memory allocation failed: " << e.what() << endl;
    }
    catch (const exception& e) {
        cerr << "Error: " << e.what() << endl;
        return 1;
    }

    fout.close();
    cout << "Experiment completed. Results saved to results.txt\n";
    return 0;
```

```
}
```