ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № Х

«Название лабораторной работы (Вашего задания)»

Выполнил работу

Кек Герман

Академическая группа №J3112

Принято

Должность, звание Фамилия Имя преподавателя

Санкт-Петербург

2024

**Структура отчёта:**

1. Введение
2. Теоретическая подготовка
3. Реализация

В этом разделе вам необходимо описать процесс выполнения работы, что вы сделали и какие этапы при этом выполняли, выжимки из кода, библиотеки и особенности реализации. Важно, не бывает 2 этапов выполнения задачи, Начали – закончили.

1. Экспериментальная часть

В этом разделе вам необходимо привести результаты работы вашего алгоритма, с таблицами и графиками, демонстрирующими выполнения алгоритма с различными условиями и наборами данных. Оценивается производительность и сравниваются результаты с теоретическими оценками.

Подсчёт по памяти (только для циклов и сложных структур) – как в лабораторной работе №2.

Подсчёт асимптотики (только для циклов и сложных структур) – как в лабораторной работе №3.

График зависимости времени от числа элементов. Пример выполнения:

Согласно требованиям моего варианта, на вход к моему алгоритму подаётся до 25 элементов. Теоретически заданная сложность задачи составляет O(2^N) и более. Для тестирования алгоритма была собрана статистика, приведенная в таблице №\*.

1. Заключение

6. Приложение

1. Введение.
   1. Цель работы — разработка и реализация алгоритма для нахождения минимального покрытия множества.
   2. Задачи:
      1. Определить минимальное количество множеств из заданного набора sets, которые покрывают все элементы из множества universe.
      2. Использовать комбинаторный способ со сложностью алгоритма не меньше O(2^N).
      3. Провести экспериментальное тестирование и оценить производительность.
2. Теоретическая подготовка.
   1. Типы переменных:
      1. vector — динамический массив, позволяющий хранить последовательности элементов переменного размера.
      2. set — ассоциативный контейнер для хранения уникальных элементов.
      3. int — целочисленный тип данных.
   2. Алгоритмы:
      1. Перебор всех комбинаций подмножеств — метод полного перебора всех возможных подмножеств заданного множества подмножеств sets.
      2. Алгоритм проверки покрытия множества — алгоритм, который проверяет, покрывает ли объединение элементов текущей комбинации подмножеств sets все элементы множества universe.
      3. Жадный подход к обновлению наилучшего покрытия — метод, который сохраняет наименьшее покрытие из всех проверенных комбинаций, обновляя минимальное покрытие, когда находится комбинация меньшего размера, покрывающая universe.
3. Реализация.
   1. Этапы выполнения задачи:
      1. **Инициализация данных.** Определил начальные множества sets и universe. Ввел вспомогательные переменные min\_size для хранения минимального размера покрытия и best для сохранения наилучшей комбинации подмножеств.
      2. **Реализация функции isCover для проверки покрытия.** Реализовал алгоритм, который проверяет покрытие множества universe путем поэлементного удаления. Для каждой комбинации подмножеств элементы этих подмножеств удаляются из universe. Если universe становится пустым, функция возвращает true, иначе – false.
      3. **Генерация комбинаций подмножеств.** Реализован переборный алгоритм для получения всех возможных комбинаций подмножеств. Этот алгоритм последовательно генерирует комбинации подмножеств, начиная с минимальных по размеру и постепенно увеличивая их размер до полного набора. Такой подход позволяет прервать выполнение цикла при нахождении первого подходящего покрытия для множества universe, что в некоторых случаях ускоряет работу алгоритма.
      4. **Вывод результата.**
4. Экспериментальная часть.
   1. Подсчёт по памяти:
      1. Функция isCover:
         1. int index – 4 байта
         2. int item – 4 байта
      2. Функция nextCombination:
         1. int k – 4 байта
         2. int j – 4 байта
      3. Функция findCover:
         1. int n – 4 байта
         2. int min\_size – 4 байта
         3. vector<set<int>> best – 24 + (32 + m1 \* 4) + (32 + m2 \* 4) + … + (32 + mn \* 4) байта (mi - количество элементов в i подмнжестве, n – количество подмножеств в best)
         4. int subset\_size – 4 байта
         5. int index – 4 байта
         6. vector<set<int>>: sets - 24 + байта (mi - количество элементов в i подмнжестве, n – количество подмножеств в sets)
         7. set<int> universe – 24 + (n \* 4) байта (n – количество элементов в universe
   2. Подсчёт асимптотики:
      1. Цикл по subset\_size – проходит от 1 до n => O(n) (n – количество элементов в sets
      2. Генерация комбинаций – перебирает все вариации подмножеств sets => O()
      3. isCover – проверка покыртия, берет все элементы из всех подномжеств combination => O(subset\_size \* m) (m – количество элементов в каждом подмножестве)

Общая сложность – O()



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размер входного набора sets | 3 | 4 | 15 | 20 | 25 |
| Размер входного набора universe | 4 | 10 | 30 | 40 | 50 |
| Время выполнения программы, мс | 0 | 0 | 582 | 6019 | 1162000000 |
| O(2\*N), мс | 0 | 0 | 1070 | 1100000 | 1130000000 |
| O(3\*N), мс | 0 | 0 | 206000000 | 121000000000 | 76200000000000 |

График представляющий визуально удобный формат данных из таблицы

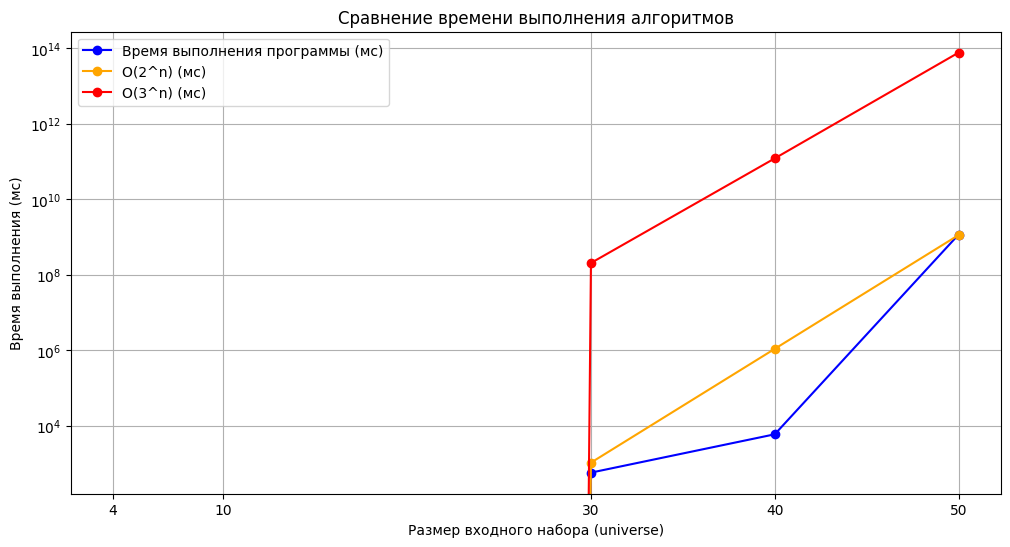
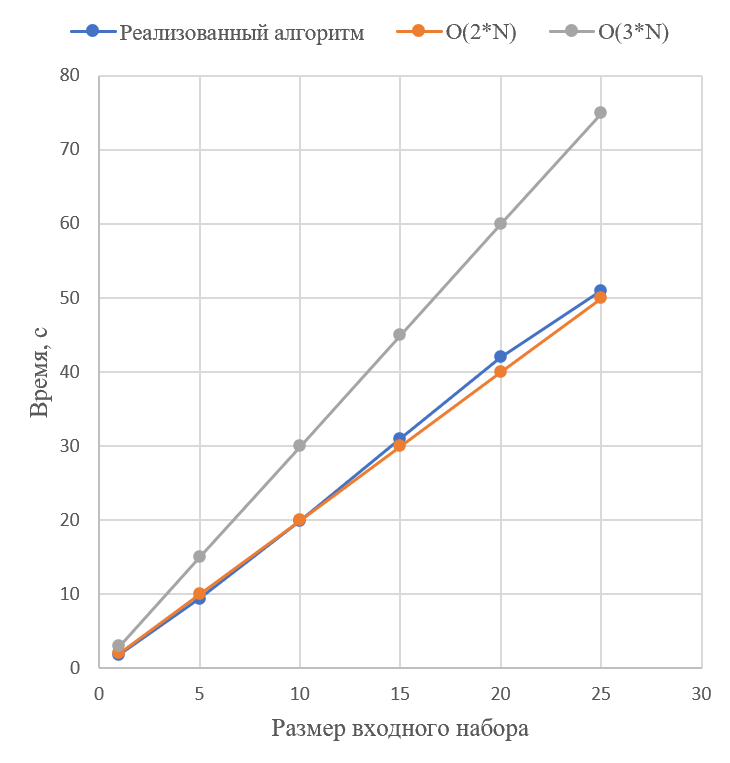
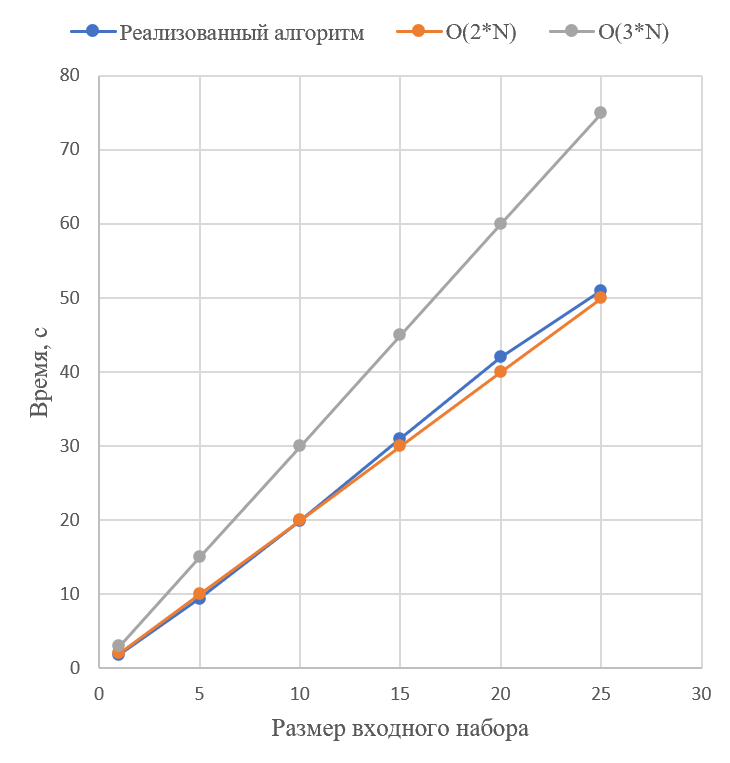


Таблица №\* - Подсчёт сложности реализованного алгоритма

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размер входного набора | 1 | 5 | 10 | 15 | 20 | 25 |
| Время выполнения программы, с | 1,8 | 9,4 | 19,9 | 32 | 42 | 51 |
| O(2\*N), с | 2 | 10 | 20 | 30 | 40 | 50 |
| O(3\*N), с | 3 | 15 | 30 | 45 | 60 | 75 |

График представляющий визуально удобный формат данных из таблицы №\* представлен на изображении №\*.



Изображение №\* - График работы алгоритма

1. Приложение:

ПРИЛОЖЕНИЕ А

Листинг кода файла main.cpp

#include <iostream>

#include <vector>

#include <set>

#include <algorithm>

#include <limits>

#include <chrono>

using namespace std;

using namespace std::chrono;

bool isCover(vector<set<int>> sets, set<int> universe, vector<int> combination) {

for (int index : combination) {

for (int item : sets[index]) {

universe.erase(item);

}

}

return universe.empty();

}

bool nextCombination(vector<int>& combination, int n) {

int k = combination.size();

for (int i = k - 1; i >= 0; i--) {

if (combination[i] < n - k + i) {

combination[i]++;

for (int j = i + 1; j < k; j++) {

combination[j] = combination[j - 1] + 1;

}

return true;

}

}

return false;

}

vector<set<int>> findCover(vector<set<int>> sets, set<int> universe) {

int n = sets.size();

int min\_size = numeric\_limits<int>::max();

vector<set<int>> best;

for (int subset\_size = 1; subset\_size <= n; subset\_size++) {

vector<int> combination(subset\_size);

for (int i = 0; i < subset\_size; i++) {

combination[i] = i;

}

while (nextCombination(combination, n)) {

if (isCover(sets, universe, combination)) {

min\_size = subset\_size;

best.clear();

for (int index : combination) {

best.push\_back(sets[index]);

}

}

if (min\_size != numeric\_limits<int>::max()) {

break;

}

}

nextCombination(combination, n);

if (isCover(sets, universe, combination)) {

min\_size = subset\_size;

best.clear();

for (int index : combination) {

best.push\_back(sets[index]);

}

}

}

return best;

}

void Testing(vector<set<int>> sets, set<int> universe) {

auto start = high\_resolution\_clock::now();

vector<set<int>> result = findCover(sets, universe);

auto end = high\_resolution\_clock::now();

auto duration = duration\_cast<milliseconds>(end - start);

cout << "[ ";

for (const set<int>& s : result) {

cout << "{ ";

for (int item : s) {

cout << item << " ";

}

cout << "} ";

}

cout << "]\n";

cout << "Тест занял: " << duration.count() << " мс" << endl;

}

int main() {

// Тест 1: 3 множеств, 4 элементов

vector<set<int>> sets1 = {{1, 2}, {2, 3}, {3, 4}};

set<int> universe1 = {1, 2, 3, 4};

cout << "Тест 1:\n";

Testing(sets1, universe1);

// Тест 2: 4 множеств, 10 элементов

vector<set<int>> sets2 = {{1, 5, 9}, {2, 5, 6, 7}, {3, 8}, {4, 10}};

set<int> universe2 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

cout << "Тест 2:\n";

Testing(sets2, universe2);

// Тест 3: 15 множеств, 30 элементов

vector<set<int>> sets3 = {

{1, 2, 3}, {4, 5, 6}, {7, 8, 9}, {10, 11, 12},

{13, 14, 15}, {16, 17, 18}, {19, 20, 21}, {22, 23, 24},

{25, 26, 27}, {28, 29, 30}, {1, 10, 20}, {5, 15, 25},

{3, 8, 12, 17}, {2, 6, 18, 24}, {11, 13, 14}

};

set<int> universe3 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10,

11, 12, 13, 14, 15, 16, 17, 18,

19, 20, 21, 22, 23, 24, 25, 26,

27, 28, 29, 30};

cout << "Тест 3:\n";

Testing(sets3, universe3);

// Тест 4: 20 множеств, 40 элементов

vector<set<int>> sets4 = {

{1, 5, 10, 15}, {2, 6, 11, 16}, {3, 7, 12, 17}, {4, 8, 13, 18},

{9, 14, 19, 20}, {21, 22, 23, 24}, {25, 26, 27, 28}, {29, 30, 31, 32},

{33, 34, 35, 36}, {37, 38, 39, 40}, {1, 6, 11, 16, 21}, {2, 7, 12, 17, 22},

{3, 8, 13, 18, 23}, {4, 9, 14, 19, 24}, {5, 10, 15, 20, 25},

{26, 27, 28, 29, 30}, {31, 32, 33, 34, 35}, {36, 37, 38, 39, 40},

{1, 2, 3, 4, 5}, {15, 25, 35, 30, 20}

};

set<int> universe4 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,

15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,

27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38,

39, 40};

cout << "Тест 4:\n";

Testing(sets4, universe4);

// Тест 5: 25 множеств, 50 элементов

vector<set<int>> sets5 = {

{1, 2, 3, 4, 5}, {6, 7, 8, 9, 10}, {11, 12, 13, 14, 15}, {16, 17, 18, 19, 20},

{21, 22, 23, 24, 25}, {26, 27, 28, 29, 30}, {31, 32, 33, 34, 35}, {36, 37, 38, 39, 40},

{41, 42, 43, 44, 45}, {46, 47, 48, 49, 50}, {1, 11, 21, 31, 41}, {2, 12, 22, 32, 42},

{3, 13, 23, 33, 43}, {4, 14, 24, 34, 44}, {5, 15, 25, 35, 45},

{6, 16, 26, 36, 46}, {7, 17, 27, 37, 47}, {8, 18, 28, 38, 48},

{9, 19, 29, 39, 49}, {10, 20, 30, 40, 50}, {1, 10, 20, 30, 40, 50},

{5, 15, 25, 35, 45}, {4, 14, 24, 34, 44}, {3, 13, 23, 33, 43},

{2, 12, 22, 32, 42}

};

set<int> universe5 = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,

17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,

31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,

45, 46, 47, 48, 49, 50};

cout << "Тест 5:\n";

Testing(sets5, universe5);

return 0;

}

**Вывод:** Работа успешно продемонстрировала высокую сложность алгоритмов полного перебора и их ограничения. Она дает представление о применении теоретических знаний на практике, включая расчет асимптотики и анализ потребляемой памяти. Для дальнейшего использования стоит рассмотреть оптимизационные методы для работы с большими наборами данных.