ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 4

«Минимальное покрытие множества»

Выполнил работу

Черницын Егор Артёмович

Академическая группа №J3114

Принято

Должность, звание Фамилия Имя преподавателя

Санкт-Петербург

2024

**Структура отчёта:**

1. Введение

Цель данной работы — реализация алгоритма минимального покрытия множества. Основная задача — определить подмножество, охватывающее все элементы целевого множества, с минимизацией числа подмножеств.

1. Теоретическая подготовка

Задача минимального покрытия множества требует использования комбинаторных методов для перебора всех возможных подмножеств. Алгоритм исчерпывающего перебора предполагает сложность O(2N), где N — количество подмножеств. Для представления подмножеств используется bitset и перевод чисел из двоичной системы счисления с помощью stoi, а структура unordered\_set позволяет формировать объединение выбранных множеств.

1. Реализация

Для решения задачи применена структура данных bitset, представляющая подмножества в виде битовой маски. Алгоритм последовательно перебирает все комбинации подмножеств и проверяет, перекрывают ли они элементы целевого множества.

Код написан на языке C++ и включает инициализацию битовых масок для выбора подмножеств, а также внутренний цикл, который проверяет, покрывают ли выбранные множества целевое множество universe.

1. Экспериментальная часть

Тестирование алгоритма проводилось на наборах данных от 1 до 25 элементов. Результаты представлены в таблице ниже.

Согласно требованиям моего варианта, на вход к моему алгоритму подаётся до 25 элементов. Теоретически заданная сложность задачи составляет O(2N\*N) и более. Для тестирования алгоритма была собрана статистика, приведенная в таблице №1.

Таблица №1 - Подсчёт сложности реализованного алгоритма

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размер входного набора | 1 | 10 | 19 | 21 | 23 | 25 |
| Время выполнения программы, с | 0 | 0 | 20 | 80 | 347 | 1700 |
| O(N2 \* 2N), с | 0 | 0 | 1 | 3 | 11 | 63 |
| O(N3 \* 2N), с | 0 | 0 | 10 | 53 | 295 | 1559 |

График представляющий визуально удобный формат данных из таблицы №1 представлен на изображении №1.

Изображение №1 - График работы алгоритма

1. Заключение

Алгоритм минимального покрытия множества успешно реализован и протестирован. Результаты совпадают с теоретическими оценками сложности. В будущем для повышения эффективности можно рассмотреть параллельные вычисления и оптимизацию по памяти и времени.

1. Приложения

ПРИЛОЖЕНИЕ А

Листинг кода файла main.cpp

#include <iostream>

#include <string>

#include <vector>

#include <bitset>

#include <fstream>

#include <unordered\_set>

#include <set>

#include <sstream>

#include <algorithm>

#include <cassert>

#include <ctime>

#include <cmath>

**using** **namespace** std;

// Ищем пересечение неотсортированных множеств. Функция работает за O(u \* m),

// где u - мощность universe, а m - общее кол-во уникальных элементов в подмножествах заданного множества (в условии не указано, сколько их может быть)

insert\_iterator<unordered\_set<**int**>> unordered\_set\_intersection(unordered\_set<**int**>::iterator b1, unordered\_set<**int**>::iterator e1, unordered\_set<**int**>::iterator b2, unordered\_set<**int**>::iterator e2, insert\_iterator<unordered\_set<**int**>> out) {

**while** (!(b1 == e1)) {

**if** (!(find(b2, e2, \*b1) == e2)) {

\*out = \*b1;

++out;

}

++b1;

}

**return** out;

}

// Функция cover для нахождения минимального покрытия множества

// Перебираем все двоичные числа от 0...01 до 1...1 длины мощности принимаемого массива, рассматривая эти числа как маски

// для выбора множеств из массива

// Итоговая сложность O(2^n \* u \* m \* 25), что позволяет оценить сложность алгоритма как > O(2^n \* n^3)

**short** **int** cover(vector<unordered\_set<**int**>> arr, unordered\_set<**int**> universe) { // arr: m \* 4 байт, universe: 200 байт

**int** arr\_size = arr.size(); // 4 байта

**long** **int** bin = stoi(string(arr\_size, '1'), nullptr, **2**); // O(n) - получаем битовую маску из всех единиц, 8 байт

**int** mincount = **26**; // 4 байта

// Основной цикл: перебор всех масок. O(2^n), т.к. возмножных масок как раз 2^n.

**for** (**long** **int** i = **1** ; i <= bin; ++i) {

// i - 8 байт

**short** **int** number\_of\_arrays = **0**; // Счетчик числа выбранных множеств, 2 байта

bitset<**25**> bitset = i; // Битовая маска для выбора множества, ~4 байта

unordered\_set<**int**> set\_for\_intersection = {}; // собираем цельное множество из выбранных для последующего его пересечения с universe

// Вложенный цикл: обработка каждого бита

**for** (**short** **int** j = **0**; j < **25**; ++j) {

// j - 2 байта

**if** (bitset[j] == **1** && j < arr\_size) {

//// Вставляем элементы множества в пересечение

set\_for\_intersection.reserve(arr[j].size());

set\_for\_intersection.insert(arr[j].begin(), arr[j].end());

//// O(1) - вставка в неупорядоченное множество

++number\_of\_arrays; // Увеличиваем счетчик

}

}

// Сложность этого цикла O(25), т.к вставка в unordered\_set - это O(1)

unordered\_set<**int**> intersection;

// Пересечение множества set\_for\_intersection с universe

unordered\_set\_intersection(universe.begin(), universe.end(), set\_for\_intersection.begin(), set\_for\_intersection.end(),

inserter(intersection, intersection.begin()));

// Если пересечение совпадает с universe, возвращаем число использованных множеств

**if** (intersection == universe) {

**if** (number\_of\_arrays < mincount) mincount = number\_of\_arrays;

}

}

// Возвращаем кол-во, -1 если покрытия нет

**if** (mincount != **26**) **return** mincount;

**else** **return** -**1**;

} // По памяти итого имеем: 4 \* m + 200 + 4 + 4 + 8 + 8 + 2 + 4 + 2 = 4 \* m + 232 байт

// Тестирующая функция

**void** test() {

vector<vector<unordered\_set<**int**>>> arrays = { { {**1**, **2**, **3**}, {**2**, **3**}, {**3**, **4**} }, { {**1**, **2**, **3**, **4**, **5**}, {**17**, **8**, **9**} },

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

vector<unordered\_set<**int**>> universums = { { **1**, **3** }, {**1**, **2**, **3**, **9**, **81**}, {**1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **10**, **11**, **12**, **13**, **14**, **15**, **16**, **17** } };

vector<**int**> right\_answers = { **1**, -**1**, **17** };

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

assert(cover(arrays[i], universums[i]) == right\_answers[i]);

}

}

// Главная функция

**int** main() {

setlocale(LC\_ALL, "Russian");

// Предварительное тестирование программы

test();

// Читаем данные из файла

string line;

string path{};

cin >> path;

ifstream **in**(path);

// Применяем алгоритм к заданному набору данных

vector<unordered\_set<**int**>> arrays{};

unordered\_set<**int**> universum{};

**if** (in.is\_open())

{

**bool** f = false;

**while** (getline(in, line)) {

unordered\_set<**int**> st{};

istringstream **iss**(line);

string token;

**while** (iss >> token) {

st.insert(stoi(token));

}

**if** (line.size() == **0**) f = true;

**if** (f == false) {

arrays.push\_back(st);

}

**else** {

universum.insert(st.begin(), st.end());

}

}

}

**else** cout << "Неверное наименование файла";

in.close();

// Вызов функции cover

**int** res = cover(arrays, universum);

cout << "Минимальное число множеств: " << res << endl;

}