ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 4

«Задача о покрытии множества»

Выполнил работу

Морозов Дмитрий

Академическая группа №J3110

Принято

Ментор, Вершинин Владислав

Санкт-Петербург

2024

**Структура отчёта:**

1. Введение

Цель работы: написать программу, которая будет возвращать минимальное количество множеств из набора B, которое сможет покрыть все элементы из множества А

Задачи:

­­­­­­­­­­— Решить задачу комбинаторным способом

— Проанализировать затрачиваемую память и время работы программы

— Построить диаграмму зависимости времени от входных данных

1. Теоретическая подготовка

Используемые типы данных:

— int для целых чисел, которые являются элементами множеств

— short int для 0 и 1 в составе бинарных масок

— double для замера времени выполнения программы

— std::vector для хранения списка множеств и бинарных масок

— std::unordered\_set для хранения множеств

— std::string для записи названий тестов

Используемые алгоритмы:

— Получение бинарных масок и работа с ними

Бинарная маска размером n, где n – количество множеств, помогает в переборе всех комбинаций выбора множеств. Если i-ый элемент бинарной маски равен 1, это значит, что i-ое множество из набора будет выбрано. Если i-ый элемент бинарной маски равен 0, i-ое множество из набора выбрано не будет.

— Перевод числа в двоичную систему счисления

Для перевода числа в двоичную систему счисления создаётся пустой в вектор. Далее создаётся цикл while, работающий до тех пор, пока исходное число больше 0. На каждом шаге цикла в начало вектора добавляется остаток от деления числа на 2. После этого число делится на 2 нацело. В результате в векторе хранится число в двоичной записи.

1. Реализация
   1. Анализ задачи, поиск оптимального решения, удовлетворяющего всем критериям.

На данном этапе был проведен анализ поставленной задачи. Учитывая, что решение должно быть комбинаторным, использовался алгоритм, который осуществляет полный перебор всех возможных комбинаций множеств. Это было осуществлено с помощью бинарных масок.

* 1. Написание программы
     1. Реализация бинарных масок

На данном этапе был реализован алгоритм получения бинарных масок. Сначала была написана функция, которая получает число в десятичной системе счисления и количество бит результата и возвращает число в двоичной системе счисления с лидирующими нулями. Она представлена на изображении 1. Далее была реализована функция, которая получает число и возвращает вектор из бинарных масок, количество которых равно заданному числу. Она представлена на изображении 2.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 1 – Функция для перевода числа в двоичную систему счисления с сохранением лидирующих нулейИзображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 2 – Функция для получения бинарных масок для данного числа

* + 1. Определение наименьшего набора множеств

На данном этапе реализована функция, которая принимает вектор из доступных множеств, а также множество, которое требуется покрыть. Функция представлена на изображении 3. Переменная best\_variant будет хранить индекс бинарной маски с наименьшим количеством множеств. В переменной min\_sets\_count хранится наименьшее количество множеств, которым можно покрыть заданное множество. Так как множеств не более 25, по умолчанию значение переменной 26, поэтому, если оно не изменится, будет выведена ошибка о невозможности покрыть множество. Далее идёт цикл, который перебирает все варианты выбора множеств. Результат объединения выбранных множеств хранится в new\_set. После заполнения new\_set идёт проверка, является ли заданное множество подмножеством new\_set. Если это произошло, то количество выбранных множеств сравнивается с min\_sets\_count. В самом конце вектор best заполняется минимальным по количеству набором множеств.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рисунок 3 – Функция, поиска покрытия минимального размера

* + 1. Проверка работы с помощью тестов

На данном этапе были написаны тесты для проверки корректности программы. Тесты проверяли работу для следующих случаев:

— Пример из условия

— Пустой набор множеств

— Пустое множество

— Оптимальный набор состоит из 1 элемента

— Оптимальный набор состоит из всех элементов

Реализация тестов представлена на изображении 4. В результате выводятся названия тестов, их статусы (пройден или нет), а также общее количество пройденных и непройденных тестов.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 4 – Реализация тестов

* 1. Экспериментальная часть работы

На данном этапе производилась оценка затрачиваемой памяти и асимптотики. Также проводились замеры времени работы программы для разных входных данных. Для замера времени использовалась библиотека ctime. На основе этого был создан график, для удобного анализа зависимости роста времени работы от входных данных.

* 1. Анализ проделанной работы

На данном этапе был выполнен анализ проделанной работы, удалось ли достичь необходимую цель и выполнить задачи.

1. Экспериментальная часть

Подсчёт по памяти:

Функция dec\_to\_bin, представленная на изображении 1, использует 2\*n байт. Функция binary\_masks, представленная на изображении 2, использует 2\*n\*2^n байт. Функция find\_sets, представленная на изображении 3, использует до 32\*n + 32 + 32\*n + 32 + 2\*n\*2^n = 64\*n + 64 + 2\*n\*2^n байт, где n – количество множеств.

Подсчёт асимптотики:

У нас есть 2^n вариантов выбора множеств. Для каждого набора запускается цикл из n повторений, чтобы получить все множества из набора в соответствии с двоичной маской. Остальные циклы серьёзно не влияют на асимптотику, поэтому итоговая сложность алгоритма: О(N\*2^N).

График зависимости времени от числа элементов.

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

Таблица №1 - Подсчёт сложности реализованного алгоритма

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размер входного набора | 1 | 5 | 10 | 15 | 20 | 25 |
| Время выполнения программы, с | 0,005 | 0,013 | 0,042 | 1,27 | 42 | 2554,06 |
| O(N\*2^N), с | 0,0001 | 0,0001 | 0,01 | 0,49 | 21 | 838 |
| O(2^N), с | 0,0001 | 0,0001 | 0,001 | 0,032 | 1,04 | 33,55 |

График представляющий визуально удобный формат данных из таблицы №1 представлен на изображении №5.

Изображение №5 - График работы алгоритма

Анализируя график, представленный на изображении №5, можно увидеть, что теоретическая сложность была определена корректна. Также она соответствует требованию быть не менее O(2^N).

1. Заключение

В ходе выполнения лабораторной работы мною был реализован алгоритм поиска минимального по размеру покрытия множества. Полученные результаты соответствуют с теоретическими оценками сложности алгоритма.

На примере этой работы можно заметить, что при высокой сложности алгоритма, применения его на больших данных неудобно, из-за больших затрат по времени.

1. Приложения

ПРИЛОЖЕНИЕ А

Листинг кода файла lab\_4.cpp

#include <iostream>

#include <vector>

#include <unordered\_set>

#include <Windows.h>

#include <ctime>

// Перевод числа в двоичную систему с сохранением лидирующих нулей

std::vector <short int> dec\_to\_bin(int digit, const int& size\_of\_res) {

std::vector <short int> res;

int count\_digits = 0; // Сколько цифр в двоичной записи числа

while (digit > 0) {

res.insert(res.begin(), digit % 2);

count\_digits++;

digit = digit / 2;

}

for (int i = 0; i < size\_of\_res - count\_digits; i++) { // Добавление лидирующих нулей

res.insert(res.begin(), 0);

}

return res;

}

// Получение всех бинарных масок

std::vector <std::vector <short int>> binary\_masks(const int& total\_variants) {

std::vector <std::vector <short int>> res;

int size\_of\_res = ceil(log2(total\_variants));

for (int i = 0; i < total\_variants; i++) {

res.push\_back(dec\_to\_bin(i, size\_of\_res));

}

return res;

}

std::vector <std::unordered\_set <int>> find\_sets(const std::vector <std::unordered\_set <int>>& sets, const std::unordered\_set <int>& universe) {

std::vector <std::unordered\_set <int>> best; // Наименьшее количество множеств

int best\_variant = 0;

int min\_sets\_count = 26;

int total\_sets = sets.size();

int total\_variants = std::pow(2, total\_sets);

std::vector <std::vector <short int>> masks = binary\_masks(total\_variants); // Бинарные маски

for (int i = 0; i < total\_variants; i++) {

int count\_sets = 0; // Сколько единиц (используется множеств)

std::unordered\_set <int> new\_set{};

for (int j = 0; j < total\_sets; j++) { // masks[i] - маска

if (masks[i][j] == 1) {

count\_sets++;

new\_set.insert(sets[j].begin(), sets[j].end());

}

}

bool flag = true;

for (auto m : universe) { // Проверка на покрытие множества

if (new\_set.find(m) == new\_set.end()) {

flag = false;

}

}

if (flag) {

if (count\_sets < min\_sets\_count) { // Сравнение с минимальным набором множеств

best\_variant = i;

min\_sets\_count = count\_sets;

}

}

}

for (int i = 0; i < total\_sets; i++) { // Заполнение результата

if (masks[best\_variant][i] == 1) {

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

}

}

if (min\_sets\_count == 26) {

std::cout << "[ERROR] Невозможно покрыть другими множествами";

}

return best;

}

int test\_passed = 0; // Пройденные тесты

int test\_failed = 0; // Нейпройденные тесты

void assertEqual(const std::vector <std::unordered\_set <int>>& sets, const std::unordered\_set <int>& universe,

std::vector <std::unordered\_set <int>> res, const std::string& testName) {

if (find\_sets(sets, universe) == res) {

std::cout << "[PASSED]" << testName << "\n";

test\_passed++;

}

else {

std::cout << "[NOT PASSED]" << testName << "\n";

test\_failed++;

}

}

void report() { // Отчёт по тестам

std::cout << "\nИтого тестов пройдено: " << test\_passed << "\n";

std::cout << "\nИтого тестов не пройдено: " << test\_failed << "\n";

}

void run\_tests() { // Запуск тестов

assertEqual({ {1, 2}, {2, 3}, {3, 4} }, { 1, 2, 3 }, { {1, 2}, {3, 4} }, "Тест: Пример из условия");

assertEqual({}, { 1, 2, 3 }, {}, "Тест: Пустой набор множеств");

assertEqual({ {1, 2}, {2, 3}, {3, 4} }, {}, {}, "Тест: Пустое множество");

assertEqual({ {1, 2}, {1, 2, 3}, {3, 4} }, { 1, 2, 3 }, { {1, 2, 3} }, "Тест: Оптимальный набор состоит из 1 элемента");

assertEqual({ {1}, {2}, {3}, }, { 1, 2, 3 }, { {1}, {2}, {3} }, "Тест: Оптимальный набор состоит из всех элемента");

report();

}

int main() {

std::clock\_t start = std::clock();

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

for (auto i : find\_sets({ {1, 2}, {2, 3}, {3, 4} }, {1, 2, 3})) {

for (auto j : i) {

std::cout << j << " ";

}

std::cout << "\n";

}

run\_tests();

std::clock\_t end = std::clock();

double duration = double(end - start) / CLOCKS\_PER\_SEC;

std::cout << "Время выполнения: " << duration << " секунд" << std::endl;

return 0;

}