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

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Построение и анализ алгоритмов»

Тема: Поиск с возвратом

Студентка гр. 3343 Гельман П.Е.

Преподаватель Жангиров Т.Р.

Санкт-Петербург

2025

Цель работы.

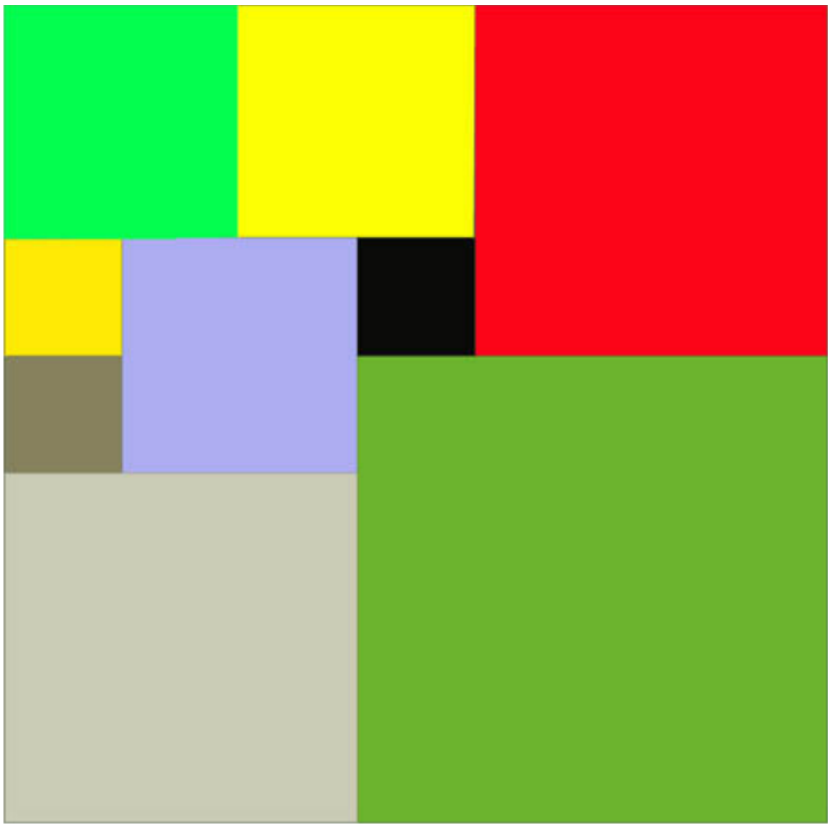
Цель данной лабораторной работы состоит в изучении бэктрекинга и разработке программы, решающей задачу с помощью алгоритма поиска с возвратом.

Задание.

Вариант 1и.

У Вовы много квадратных обрезков доски. Их стороны (размер) изменяются от 1 до N−1, и у него есть неограниченное число обрезков любого размера. Но ему очень хочется получить большую столешницу - квадрат размера N. Он может получить ее, собрав из уже имеющихся обрезков(квадратов).

Например, столешница размера 7×7 может быть построена из 9 обрезков.



Внутри столешницы не должно быть пустот, обрезки не должны выходить за пределы столешницы и не должны перекрываться. Кроме того, Вова хочет использовать минимально возможное число обрезков.

Входные данные

Размер столешницы – одно целое число N $(2 \le N \le 20)$.

Выходные данные

Одно число K, задающее минимальное количество обрезков(квадратов), из которых можно построить столешницу(квадрат) заданного размера N. Далее должны идти K строк, каждая из которых должна содержать три целых числа x,y и w, задающие координаты левого верхнего угла ($1 \le x,y \le N$) и длину стороны соответствующего обрезка (квадрата).

Пример входных данных

7

Соответствующие выходные данные

9

1 1 2

1 3 2

3 1 1

4 1 1

3 2 2

5 1 3

4 4 4

1 5 3

3 4 1

Выполнение работы.

1. Структура Square представляет собой квадрат, хранит координаты x и y, а также длину одной стороны фигуры.

2. Класс Board – класс, представляющий столешницу. Его поля: N-размер одной стороны стола, board – вектор векторов с типом int (матрица), где каждая ячейка является как бы клеткой стола, вектор квадратов result, в котором будут храниться те квадраты, которыми можно минимально заполнить стол, вектор квадратов temporary – хранит текущее разбиение стола на квадраты, нужен для сравнения и поиска минимального разбиения.

Методы:

- explicit Board(int N) – конструктор класса, принимает на вход размер стола, инициализирует board нулями.
- vector<Square> numberDivisor() - деление стола на квадраты в зависимости от N. Создан для оптимизации работы алгоритма. Если N четное, то всегда минимальное число квадратов – 4, если N делится на 3 без остатка – 6 квадратов, если на 5 – 8 квадратов, если на 7 – 9 квадратов.
- std::vector<std::vector<int>>& fillSquare(Square square) – метод, который заполняет переданный ему квадрат на доске единицами.
- bool isSquareEmpty(Square square) const – проверяет, пуста ли та квадратная область, которую мы хотим выбрать в качестве нового квадрата для заполнения стола.
- Square findEmptySquare () – находит на доске самый большой возможный пустой квадрат и возвращает его.
- std::vector<std::vector<int>>& deleteLastSquare(Square square) – удаляет последний квадрат, ставит все ячейки в 0, возвращает измененный стол.
- bool isTableFull() const – проверяет, заполнена ли столешница квадратами.
- std::vector<std::vector<int>>& cutSquare(Square square) - уменьшение размера последнего квадрата на 1, для дальнейшей проверки.

Итеративно ставит в 0 самую правую и самую нижнюю границы квадрата, тем самым уменьшая длину по вертикали и по горизонтали на 1.

- void printBoard() const – метод, выводящий в консоль ячейки стола.
- void fillEvenSquares() - заполняет стол для четного N четырьмя квадратами.
- void fillDivisibleByThreeSquares() - заполняет стол для N, кратного 3, шестью квадратами
- void fillDivisibleByFiveSquares() - заполняет стол для N, кратного 5, восемью квадратами.
- void fillDivisibleBySevenSquares() - заполняет стол для N, кратного 7, девятью квадратами.
- vector<Square> backtracking() – метод, реализующий итеративный бэктрекинг:

Переменная start – флаг, который нужен для работы алгоритма впервые. После первой итерации становится false. Запускается цикл, пока размер временного вектора квадратов непустой или программа впервые заполняет его. Вложенный цикл – пока столешница не заполнена. Проверяется, не больше ли размер temporary, чем размер result. Если это так, то выходим из этого цикла, так как нет смысла рассматривать дальше временный вектор, который больше по длине, чем результирующий. Иначе с помощью findEmptySquare() находим наибольший пустой квадрат, доступный сейчас на столе. Заполняем этот квадрат единицами и добавляем во временный вектор. После заполнения вектора векторов проверяем, меньше ли размер temporary размера result, если да, то новый result – это temporary. Далее удаляем последний добавленный квадрат в temporary, так как, если мы далее будем уменьшать его стороны, то количество квадратов станет только больше. Затем идет цикл, пока размер временного вектора ненулевой и длина у последнего квадрата этого вектора равна 1. В этом цикле продолжаем удаление, так как уменьшить сторону единичного

квадрата можно только до 0, что не повлияет на решение. Если после этого остались квадраты в temporary, то уменьшаем стороны последнего на единицу. И цикл запускается заново, и ищется новое заполнение квадратами.

Оптимизации:

1. Все числа, кратные 2, 3, 5 или 7 имеют ту же минимальную расстановку квадратов, что и квадраты с длинами 2, 3, 5, 7 только увеличенную в N/2, N/3, N/5, N/7 раз. Для квадрата со стороной 2 минимальная расстановка это 4 квадрата. Эта расстановка минимальна для всех четных длин столов, но в случае нечетных чисел, кратных 3, 5, 7, не делящихся на 2, верхний левый квадрат будет длины k, а левый нижний и правый верхний длины k - 1 соответственно. Остальное поле будет заполняться минимальным количеством квадратов.

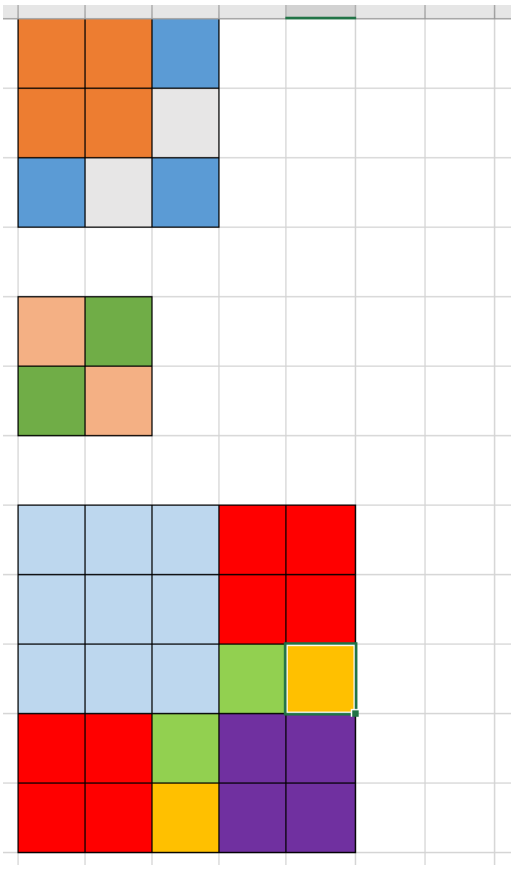


Рисунок 1 – Заполнение квадратов

По этой логике можно отследить, что в минимальном наборе для любой длины квадрата всегда присутствуют 3 основных квадрата.

Соответственно, заполнение любой столешницы начинается с заполнения тремя квадратами с длинами (N / 2 + 1), (N / 2), (N / 2).

2. Вместо того, чтобы каждый раз ставить и стирать квадраты в программе урезается правая и нижняя грани квадрата. Таким образом, его площадь уменьшается на 1, и рассматриваются другие варианты заполнения.

Для каждого вызова функции хранится вектор векторов точек столешницы и вектора квадратов длиной не больше n.

Следовательно, сложность по памяти O(N^2+2N). Сложность алгоритма по времени - O((N)^2).

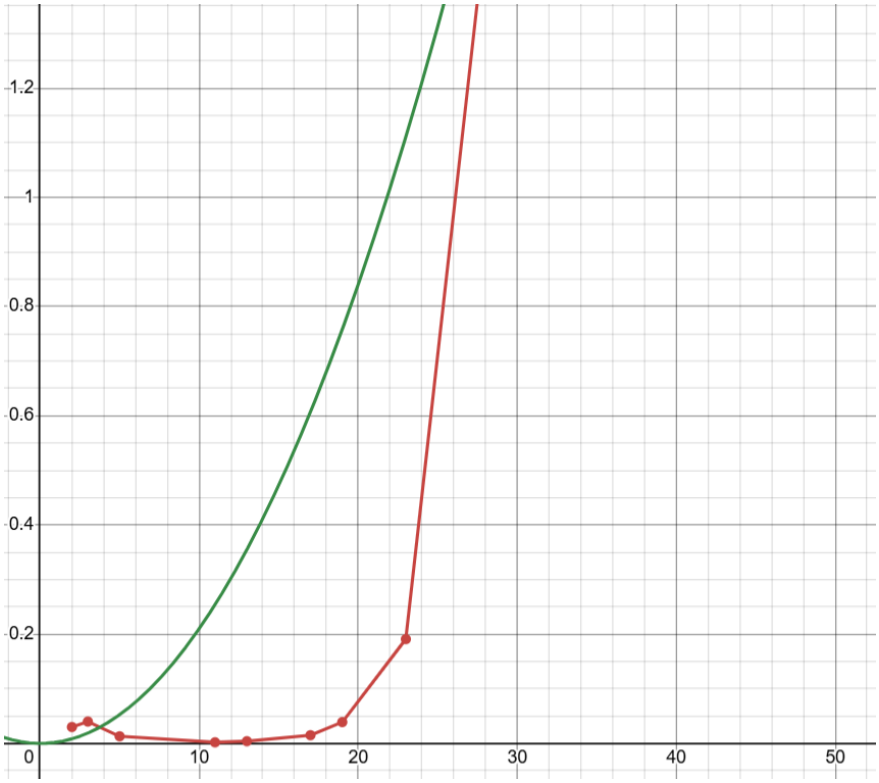


Рисунок 2 – Оценка по времени

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	N=2	4	Верно

		1 2 1 1 1 1 2 2 1 2 1 1	
2.	N=4	4 1 3 2 1 1 2 3 3 2 3 1 2	Верно
3.	N=9	6 1 1 6 1 7 3 4 7 3 7 7 3 7 1 3 7 4 3	Верно
4.	N=7	9 1 1 4 1 5 3 5 1 3 4 5 1 4 6 2 6 6 2 5 4 2 7 4 1 7 5 1	Верно
5.	N=11	11 1 1 6 7 1 5 1 7 5 6 7 3 6 10 2 7 6 1 8 6 1 8 10 1 8 11 1 9 6 3 9 9 3	Верно
6.	N=23	13 1 1 12 13 1 11 1 13 11 12 13 2 12 15 5 12 20 4 13 12 1 14 12 3 16 20 1 16 21 3 17 12 7	Верно

		17 19 2 19 19 5	

Выводы.

В ходе лабораторной работы был реализован алгоритм поиска с возвратом, проанализирована его временная сложность и сложность по памяти.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include "Board.hpp"
#include <ctime>
#include <limits>
#include <cmath>

bool isPrime(int N){
    for (int i = 2; i < sqrt(N); ++i) {
        if (N % i == 0) return false;
    }
    return true;
}

int main(){
    int N;
    while (true) {
        if (!(cin >> N) || N < 2 || N > 30) {
            cout << "incorrect input\n";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        } else {
            break;
        }
    }
    Board Board1 = Board(N);
    vector<Square> res;
    clock_t start = clock();

    if (N % 2 == 0 || N % 3 == 0 || N % 5 == 0 || N % 7 == 0){
        res = Board1.numberDivisor();
    }else {
        if (isPrime(N)) {
            Board1.board = Board1.fillSquare({0, 0, N / 2 + 1});
            Board1.board = Board1.fillSquare({0, N / 2 + 1, N / 2});
            Board1.board = Board1.fillSquare({N / 2 + 1, 0, N / 2});
            res = Board1.backtracking();
            res.insert(res.begin(), {0, N / 2 + 1, N / 2});
            res.insert(res.begin(), {N / 2 + 1, 0, N / 2});
            res.insert(res.begin(), {0, 0, N / 2 + 1});

        }
    }
    clock_t stop = clock();
    double duration = double(stop - start) / CLOCKS_PER_SEC;

    cout << res.size() << '\n';
    for (auto & square : res) {
        cout << square.x + 1 << ' ' << square.y + 1 << ' ' <<
             square.side << '\n';
    }
    cout << duration << " seconds" << endl;

    return 0;
```

```
}
```

Название файла: Board.hpp

```
#ifndef BOARD_HPP
#define BOARD_HPP

#include <vector>
#include <iostream>
#include <algorithm>

using namespace std;

struct Square {
    int x, y, side;
};

class Board {
public:
    int N;
    vector<vector<int>> board;
    vector<Square> result;
    vector<Square> temporary;

    explicit Board(int N);
    vector<Square> numberDivisor();
    vector<Square> backtracking();

    std::vector<std::vector<int>>& fillSquare(Square square);
    bool isSquareEmpty(Square square) const;
    Square findEmptySquare();
    std::vector<std::vector<int>>&      deleteLastSquare(Square
square);
    bool isTableFull() const;
    std::vector<std::vector<int>>& cutSquare(Square square);
    void printBoard() const;

    void fillEvenSquares();
    void fillDivisibleByThreeSquares();
    void fillDivisibleByFiveSquares();
    void fillDivisibleBySevenSquares();
```

```
};

#endif
```

Название файла: Board.cpp

```
#include "Board.hpp"
#include <algorithm>

Board::Board(int N) : N(N) {
    board.resize(N, std::vector<int>(N, 0));
    result.resize(N);
}

vector<Square> Board::numberDivisor() {
    result.clear();

    if (N % 2 == 0) {
        fillEvenSquares();
    } else if (N % 3 == 0) {
        fillDivisibleByThreeSquares();
    } else if (N % 5 == 0) {
        fillDivisibleByFiveSquares();
    } else if (N % 7 == 0) {
        fillDivisibleBySevenSquares();
    }

    return result;
}

void Board::fillEvenSquares() {
    int half = N / 2;
    for (int i = 1; i <= 4; ++i) {
        int x = (i <= 2) ? 0 : half;
        int y = (i % 2 == 0) ? 0 : half;
        result.push_back({x, y, half});
        board = fillSquare({x, y, half});
    }
    cout << "Divisible by 2\n";
    printBoard();
```

```
    }

    void Board::fillDivisibleByThreeSquares() {
        result.push_back({0, 0, (N * 2) / 3 });                    //*
*   *
        result.push_back({0, (N * 2) / 3, N / 3 });                //*
*   *
        result.push_back({N / 3, (N * 2) / 3, N / 3 });            //*
*   *
        result.push_back({(N * 2) / 3, (N * 2) / 3, N / 3 });
        result.push_back({(N * 2) / 3, 0, N / 3 });
        result.push_back({(N * 2) / 3, N / 3, N / 3 });

        for (const auto& square : result) {
            board = fillSquare(square);
        }
        cout << "Divisible by 3\n";
        printBoard();
    }

    void Board::fillDivisibleByFiveSquares() {
        result.push_back({0, 0, (N * 3) / 5 });
        result.push_back({0, (N * 3) / 5, (N * 2) / 5 });
        result.push_back({(N * 3) / 5, 0, (N * 2) / 5 });
        result.push_back({(N * 3) / 5, (N * 3) / 5, (N * 2) / 5 });
        result.push_back({(N * 2) / 5, (N * 3) / 5, N / 5 });
        result.push_back({(N * 2) / 5, (N * 4) / 5, N / 5 });
        result.push_back({(N * 3) / 5, (N * 2) / 5, N / 5 });
        result.push_back({(N * 4) / 5, (N * 2) / 5 , N / 5 });

        for (const auto& square : result) {
            board = fillSquare(square);
        }
        cout << "Divisible by 5\n";
        printBoard();
    }

    void Board::fillDivisibleBySevenSquares() {
        result.push_back({0, 0, (N * 4) / 7 });
```

```
        result.push_back({0, (N * 4) / 7, (N * 3) / 7 });
        result.push_back({(N * 4) / 7, 0, (N * 3) / 7 });
        result.push_back({(N * 3) / 7, (N * 4) / 7, N / 7});
        result.push_back({(N * 3) / 7, (N * 5) / 7, (N * 2) / 7});
        result.push_back({(N * 5) / 7, (N * 5) / 7, (N * 2) / 7});
        result.push_back({(N * 4) / 7, (N * 3) / 7, (N * 2) / 7});
        result.push_back({(N * 6) / 7, (N * 3) / 7, N / 7});
        result.push_back({(N * 6) / 7, (N * 4) / 7, N / 7});

        for (const auto& square : result) {
            board = fillSquare(square);
        }
        cout << "Divisible by 7\n";
        printBoard();
    }

    std::vector<std::vector<int>>& Board::fillSquare(Square square)
{
        for (int i = square.x; i < square.x + square.side; ++i) {
            for (int j = square.y; j < square.y + square.side; ++j)
{
                board[i][j] = 1;
            }
        }
        return board;
    }

    bool Board::isSquareEmpty(Square square) const {
        for (int i = square.x; i < square.x + square.side; ++i) {
            for (int j = square.y; j < square.y + square.side; ++j)
{
                if (board[i][j] != 0) return false;
            }
        }
        return true;
    }

    Square Board::findEmptySquare() {
        for (int i = 0; i < N; ++i) {
```

```
            for (int j = 0; j < N; ++j) {
                if (board[i][j] == 0) {
                    int side = N - std::max(i, j);
                    while (!isSquareEmpty({i, j, side})) {
                        --side;
                    }
                    return {i, j, side};
                }
            }
        }
        return {0, 0, 0};
    }

    std::vector<std::vector<int>>&   Board::deleteLastSquare(Square
square) {
        for (int i = square.x; i < square.x + square.side; ++i) {
            for (int j = square.y; j < square.y + square.side; ++j)
{
                board[i][j] = 0;
            }
        }
        return board;
    }

    bool Board::isTableFull() const {
        for (int i = 0; i < N; ++i) {
            for (int j = 0; j < N; ++j) {
                if (board[i][j] == 0) return false;
            }
        }
        return true;
    }

    std::vector<std::vector<int>>& Board::cutSquare(Square square) {
        for (int i = square.x; i < square.x + square.side; ++i) {
            for (int j = square.y; j < square.y + square.side; ++j)
{
                if (i == square.x + square.side - 1 || j == square.y
+ square.side - 1) {
```

```
                    board[i][j] = 0;
                }
            }
        }
        return board;
    }

    void Board::printBoard() const {
        for (int i = 0; i < N; ++i) {
            for (int j = 0; j < N; ++j) {
                std::cout << board[i][j] << "  ";
            }
            std::cout << '\n';
        }
        std::cout << '\n';
    }

    vector<Square> Board::backtracking() {
        bool start = true;

        while (!temporary.empty() || start) {
            start = false;
            while (!isTableFull()) {
                if (temporary.size() >= result.size()) break;
                Square temp = findEmptySquare();
                cout << "Found square's size: " << temp.side << endl;
                board = fillSquare(temp);
                temporary.push_back(temp);
                cout << "Filling the table while !isTableFull or
temporary.size() < result.size()\n";
                printBoard();
            }
            if (temporary.size() < result.size()) {
                cout << "New found number of temporary squares: " <<
temporary.size() << endl;
                result = temporary;
            }
            board = deleteLastSquare(temporary.back());
            temporary.pop_back();
```

```
            cout << "Delete last square from board\n";
            printBoard();

            while (!temporary.empty() && temporary.back().side == 1)
{
                board = deleteLastSquare(temporary.back());
                temporary.pop_back();
                cout << "Delete squares from temporary while it's
side == 1\n";
                printBoard();
            }

            if (!temporary.empty()) {
                board = cutSquare(temporary.back());
                temporary.back().side--;
                cout << "Cutting the last square\n";
                printBoard();
            }
        }
        cout << "Temporary is empty\n";

        return result;
}
```