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

Студент гр. 8304 _____ Щука А. А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2020

Цель работы.

Ознакомиться с алгоритмом поиска с возвратом, научиться оценивать временную сложность алгоритма и применять его для решения задач.

Постановка задачи.

Вариант 1р. Рекурсивный бэктрекинг. Поиск решения за разумное время (меньше 2 минут) для $2 \leq N \leq 40$.

Входные данные:

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

Выходные данные:

Одно число задающее минимально 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 $(1 \leq x, y \leq N)$ и длину стороны соответствующего обрезка (квадрата).

Описание алгоритма.

Алгоритм разбиения:

1) Если квадрат оптимальный (сторона делится на 2, 3, 5), тогда алгоритм за O(1) вычисляет разбиение и завершает работу.
2) Если квадрат неоптимальный, запускается оптимизированный алгоритм разбиения.
3) Исходный квадрат делится на 3 квадрата и на неполный квадрат. Разбиение представлено на рис. 1.

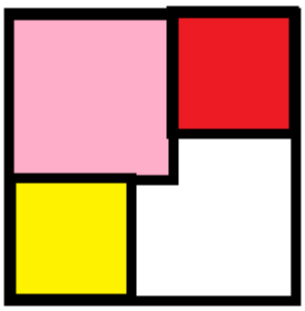

Рисунок 1 – Разбиение квадрата

4) Для неполного квадрата запускается бэктрекинг и находится минимальное разбиение.

Анализ алгоритма.

Для оптимальных квадратов алгоритм находит решение за O(1), для неоптимальных – площадь квадрата для бэктрекинга примерно равна ¼ исходной площади, что существенно сокращает время работы. Точную оценку времени работы бэктрекинга, из-за того, что он рекурсивный и содержит циклы, дать сложно. Грубая оценка– экспоненциальное время от размера неполного квадрата. Алгоритм дольше всего работает для квадратов, у которых длина стороны – простое число. Для квадрата с длиной стороны, равной 37, алгоритм находит решение менее, чем за 3 секунды, что соответствует временному промежутку в 2 минуты. Алгоритм использует O(n^2) памяти.

Описание функций и СД.

Для решения задачи был реализован класс TableTop.

Класс содержит методы вывода на экран промежуточных решений, минимального числа квадратов, результата решения.

Промежуточные решения хранятся в двумерном массиве.

Функция бэктрекинга:

void TableTop::backtracking(int length, int x, int y)

принимает на вход длину квадрата, и координаты левого верхнего угла, возвращаемое значение отсутствует. Функция записывает промежуточные данные и результат в поля класса.

void TableTop::startBacktracking()

Единственный публичный метод, который запускает бэктрекинг, применяя оптимизацию разбиением.

void TableTop::paintSquare(int x, int y, int length)

Метод раскрашивает квадрат в текущий цвет и меняет цвет.

bool TableTop::findAvaibleCoord(int x, int y, int& savedX, int& savedY)

Метод находит координаты некрашеной клетки.

bool TableTop::canPaintSquare(int x, int y, int length)

Метод проверки на возможность закрасить квадрат.

void TableTop::clearSquare(int x, int y, int len)
Отчищает массив-буфер.

void TableTop::checkMinSquare()
Метод сохранения результата минимального разбиения

Методы для вывода информации на экран
void TableTop::printSquare(Square** square)
void TableTop::writeRes()

Спецификация программы.

Программа предназначена для нахождения минимального способа разбиения квадрата на меньшие квадраты. Программа написана на языке C++. Входными данными является число N (сторона квадрата), выходными – минимальное количество меньших квадратов и K строк, содержащие координаты левого верхнего угла и длину стороны соответствующего квадрата. Результат работы программы представлен на рис. 2.

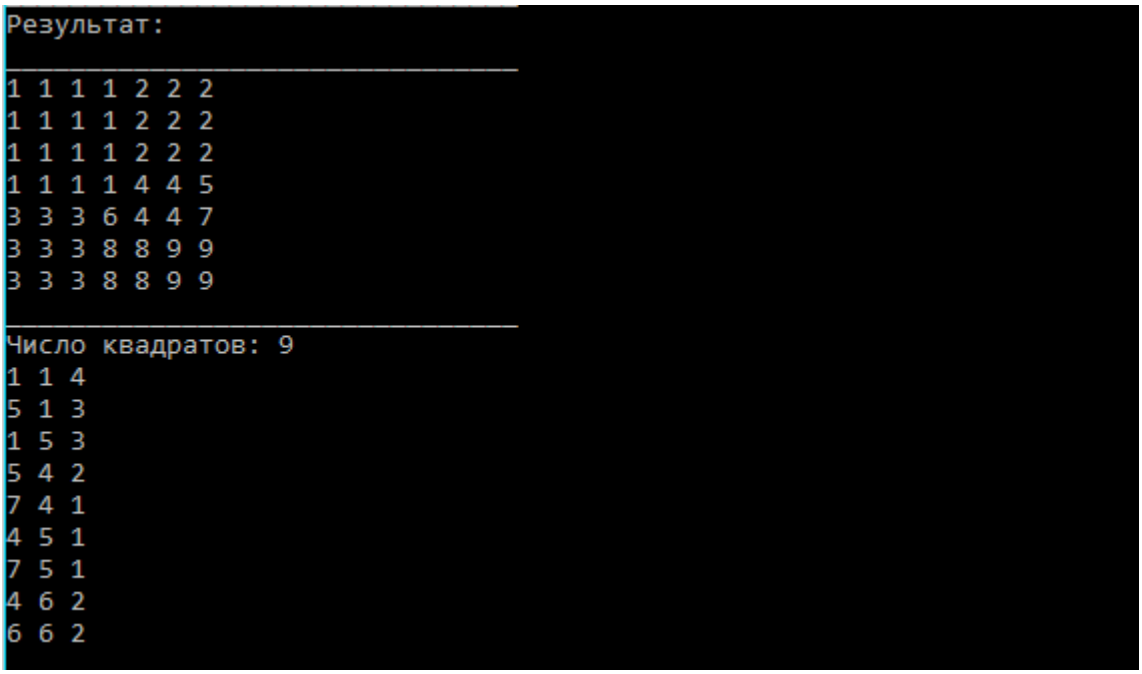

Рисунок 2 – Результат работы программы

Выводы.

В ходе выполнения лабораторной работы был реализован алгоритм рекурсивного бэктрекинга, дана оценка времени работы алгоритма, а также были получены навыки решения задач с помощью поиска с возвратом.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД ПРОГРАММЫ

main.cpp.

```
#include <iostream>
#include <Windows.h>
#include <ctime>

#include "TableTop.h"


bool isOptimalLength(int n);
void optimalSolution(int n);


int main() {
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);

    std::cout << "Введите длину строны квадрата [2; 40]: ";
    int n = 0; //Длина стороны квадрата
	std::cin >> n;
	if (std::cin.bad()) {
		std::cout << "\nОшибка ввода";
	 exit(1);
	}
	if (n < 2 || n > 40) {
		std::cout << "\nОшибка ввода";
	 exit(1);
	}

    if (isOptimalLength(n)) { //Если можно дать ответ сразу - вызываем соответствующую функцию
		optimalSolution(n);
	}
	else { //Иначе - вызываем менее производительную функцию с бэктрекингом
		TableTop tableTop(n);
      //   auto startTime = clock();
		tableTop.startBacktracking();
      //  auto endTime = clock();
      //  std::cout << endTime - startTime << std::endl;
	  }

	  std::cout << std::endl;
	  return 0;
}


bool isOptimalLength(int n) {
	if (n % 2 == 0 || n % 3 == 0 || n % 5 == 0) {
		return true;
	}
	return false;
}


void optimalSolution(int n) {
    if (n % 2 == 0) {
```

```
        std::cout << "\nЧисло квадратов: 4\n"; //Получится 4 одинаковых квадрата
              std::cout << "1 1 " << n/2 << std::endl;
              std::cout << 1 + n/2 << " 1 " << n/2 << std::endl;
              std::cout << "1 " << 1 + n/2 << " " << n/2 << std::endl;
              std::cout << 1 + n/2 << " " << 1 + n/2 << " " << n/2 << std::endl;
       }
    else if (n % 3 == 0) {
        std::cout << "\nЧисло квадратов: 6\n" << std::endl; // Получится квадрат размером 2/3
        std::cout << "1 1 " << 2 * n / 2 << std::endl;      // от размера заданного и 5 квадратов
размером 1/3
              std::cout << 1 + 2 * n / 2 << " 1 " << n / 2 << std::endl;
              std::cout << "1 " << 1 + 2 * n / 2 << " " << n / 2 << std::endl;
              std::cout << 1 + 2 * n / 2 << " " << 1 + n / 2 << " " << n / 2 << std::endl;
              std::cout << 1 + n / 2 << " " << 1 + 2 * n / 2 << " " << n / 2 << std::endl;
              std::cout << 1 + 2 * n / 2 << " " << 1 + 2 * n / 2 << " " << n / 2 << std::endl;
       }
    else if (n % 5 == 0) {
        std::cout << "\nЧисло квадратов: 8\n" << std::endl; // Получится квадрат размером 3/5 от
        std::cout << "1 1 " << 3 * n / 5 << std::endl;      // размера заданного, 3 квадрата 2/5 и
4 квадрата 1/5
              std::cout << 1 + 3 * n / 5 << " 1 " << 2 * n / 5 << std::endl;
              std::cout << "1 " << 1 + 3 * n / 5 << " " << 2 * n / 5 << std::endl;
              std::cout << 1 + 3 * n / 5 << " " << 1 + 3 * n / 5 << " " << 2 * n / 5 << std::endl;
              std::cout << 1 + 2 * n / 5 << " " << 1 + 3 * n / 5 << " " << n / 5 << std::endl;
              std::cout << 1 + 2 * n / 5 << " " << 1 + 4 * n / 5 << " " << n / 5 << std::endl;
              std::cout << 1 + 3 * n / 5 << " " << 1 + 2 * n / 5 << " " << n / 5 << std::endl;
              std::cout << 1 + 4 * n / 5 << " " << 1 + 2 * n / 5 << " " << n / 5 << std::endl;
       }
}
```

tabletop.cpp

```
#include "TableTop.h"


TableTop::TableTop(int length)
{
       this->length = length;

       bufSquare = new Square*[length];
    minSquare = new Square*[length];

    for (auto i = 0; i < length; ++i) {
        bufSquare[i] = new Square[length];
        minSquare[i] = new Square[length];
       }

       minSquareNum = length * length;
       colorCount = 0;
       squareCount = 0;
}


TableTop::~TableTop()
{
    for (auto i = 0; i < length; ++i) {
        delete [] bufSquare[i];
        delete [] minSquare[i];
    }
```

```
    delete [] bufSquare;
    delete [] minSquare;
}


void TableTop::startBacktracking()
{
       paintSquare(0, 0, length/2 + 1);
       paintSquare(length / 2 + 1, 0, length / 2);
    paintSquare(0, length / 2 + 1, length / 2); //оптимизация алгоритма, оставляем 1/4 часть квадрата

       backtracking(length/2, length/2, length/2);
       writeRes();
}


void TableTop::paintSquare(int x, int y, int length)
{
    ++squareCount;
    for (auto i = y; i < y + length; ++i) {
        for (auto j = x; j < x + length; ++j) {
                    bufSquare[i][j].size = length;
                    bufSquare[i][j].number = squareCount;
             }
       }

       colorCount++;
       if (this->length <= 10) {
              std::cout << "Покраска " << colorCount << std::endl;
              printSquare(bufSquare);
       }
}


void TableTop::printSquare(Square** square)
{
       std::cout << "____________________________________" << std::endl;
    for (auto i = 0; i < length; ++i) {
        for (int j = 0; j < length; ++j) {
                    std::cout << square[i][j].number << " ";
             }
             std::cout << std::endl;
       }
       std::cout << "____________________________________" << std::endl;
}


void TableTop::backtracking(int length, int x, int y)
{
       if (squareCount >= minSquareNum) {
              return;
       }

       int savedX;
       int savedY;

       if (findAvaibleCoord(x, y, savedX, savedY)) {
        for (auto len = length; len > 0; --len) {
                    if (canPaintSquare(savedX, savedY, len)) {
```

```
                        backtracking(length, x, savedY);
                        clearSquare(savedX, savedY, len);
                    }
                }
                return;
        }
        checkMinSquare();
}


bool TableTop::findAvaibleCoord(int x, int y, int& savedX, int& savedY)
{
    for (auto i = y; i < length; ++i) {
        for (auto j = x; j < length; ++j) {
                    if (bufSquare[i][j].size == 0) {
                        savedX = j;
                        savedY = i;
                        return true;
                    }
                }
        }
        return false;
}


bool TableTop::canPaintSquare(int x, int y, int length)
{
        if (x + length > this->length || y + length > this->length) {
                return false;
        }

    for (auto i = y; i < y + length; ++i) {
        for (int j = x; j < x + length; ++j) {
                    if (bufSquare[i][j].size != 0) {
                        return false;
                    }
                }
        }

        paintSquare(x, y, length);

        return true;
}


void TableTop::clearSquare(int x, int y, int len)
{
    for (auto i = y; i < y + len; ++i) {
        for (int j = x; j < x + len; ++j) {
                    bufSquare[i][j].number = 0;
                    bufSquare[i][j].size = 0;
                }
        }
    --squareCount;
}


void TableTop::checkMinSquare()
{
```

```
        if (squareCount < minSquareNum) {
                minSquareNum = squareCount;

         for (auto i = 0; i < length; ++i) {
             for (int j = 0; j < length; ++j) {
                                minSquare[i][j] = bufSquare[i][j];
                        }
                }
        }
}


void TableTop::writeRes()
{
        std::cout << "Результат: " << std::endl;
        printSquare(minSquare);
    std::cout << "Число квадратов: " << minSquareNum << std::endl;

    for (auto i = 1; i <= minSquareNum; ++i) {
         for (auto y = 0; y < length; ++y) {

                        int len = 0;
             for (auto x = 0; x < length; ++x) {
                                if (minSquare[y][x].number == i) {
                                        len = minSquare[y][x].size;
                                        std::cout << x + 1 << " " << y + 1 << " " << len << std::endl;
                                        break;
                                }
                        }

                        if (len) {
                                break;
                        }
                }
        }
}
```

square.h

```
#pragma once


struct Square  //структура дл¤ хранени¤ единичного квадрата
{
    Square()
    {
         size = 0;
         number = 0;
    }

        int size;
        int number;
};
```

tabletop.h

```
#pragma once
#include "Square.h"
#include <iostream>
```

```
class TableTop
{
public:
    TableTop(int length);
    ~TableTop();
    void startBacktracking();

private:
       void paintSquare(int x, int y, int length);
       void printSquare(Square** square);
       void backtracking(int length, int x, int y);
       bool findAvaibleCoord(int x, int y, int& savedX, int& savedY);
       bool canPaintSquare(int x, int y, int length);
       void clearSquare(int x, int y, int len);
       void checkMinSquare();
       void writeRes();

private:
    Square** bufSquare;
    Square** minSquare;
    int length;
    int minSquareNum;
    int colorCount;
    int squareCount;
};
```