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. 7304 _____ Пэтайчук Н.Г.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2019

Цель работы

Исследование алгоритмов поиска с возвратом, а также методы их оптимизации для практического применения.

Постановка задачи

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 её, собрав из уже имеющихся обрезков (квадратов). 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 ($2 \leq N \leq 40$).

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

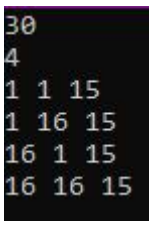
Ход работы

1. Объявление и определение структуры обрезка, содержащей в себе координаты верхнего левого угла и размер;
2. Написание ряда функций, позволяющих работать двумерным массивом как с столешницей, куда необходимо класть квадратики;
3. Создание функций, рассматривающие особые случаи, когда можно сразу дать ответ (когда сторона большого квадрата делится на 2, 3 и 5);
4. Написание функции бектрекинга (поиска с возвратом), которая ищет способ образования столешницы с помощью минимального числа обрезков, а также функции-обёртки, которая оптимизирует сам бектрекинг;
5. Написание головной функции, где будет считываться размер столешницы и выводится результат работы бектрекинга;

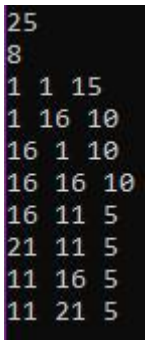
Весь исходный код программы представлен в Приложении 1.

Тестирование программы

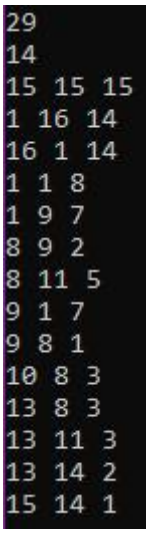
1) Было введено число 30:



2) Было введено число 25:



3) Было введено число 29:



Вывод

В ходе данной лабораторной работы были изучены алгоритм поиска с возвратом и варианты его оптимизации. Также были изучены способы применения данного алгоритма для решения практических задач.

Приложение 1: Исходный код программы

```
#include <iostream>

using namespace std;

struct Square
{
    int x;
    int y;
    int size;
};

typedef Square *square_list;

//Функции для работы с полем большого квадрата

bool is_enough_space(int **square_field, int square_size, int i, int j, int size)
{
    if ((i + size > square_size) || (j + size > square_size))
        return false;
    for (int x = i; x - i < size; x++)
        for (int y = j; y - j < size; y++)
            if (square_field[x][y] != 0)
                return false;
    return true;
}

bool is_full(int **square_field, int square_size)
{
    for (int i = 0; i < square_size; i++)
        for (int j = 0; j < square_size; j++)
            if (square_field[i][j] == 0)
                return false;
    return true;
}

void insert_square(int **square_field, Square &square)
{
    for (int i = square.x; i - square.x < square.size; i++)
        for (int j = square.y; j - square.y < square.size; j++)
            square_field[i][j] = 1;
}

void delete_square(int **square_field, Square &square)
{
    for (int i = square.x; i - square.x < square.size; i++)
        for (int j = square.y; j - square.y < square.size; j++)
            square_field[i][j] = 0;
}

void print_field(int ** square_field, int square_size)
{
    cout << "--------------" << endl;
    for (int i = 0; i < square_size; i++)
    {
        for (int j = 0; j < square_size; j++)
            cout << square_field[i][j] << " ";
        cout << endl;
    }
    cout << "--------------" << endl;
}
```

```
//-----------------------------------------------

//Функции, выполняющие разбиение квадрата на минимальное число обрезков
//Случай, сторона делится на 2
void formAnswer_multOf2(square_list &solution, int &min_count, int size)
{
    min_count = 4;
    int size_1 = size / 2;
    int coordinate_1 = size_1 + 1;
    solution[0] = {1, 1, size_1};
    solution[1] = {1, coordinate_1, size_1};
    solution[2] = {coordinate_1, 1, size_1};
    solution[3] = {coordinate_1, coordinate_1, size_1};
    return;
}

//Случай, когда сторона делится на 3
void formAnswer_multOf3(square_list &solution, int &min_count, int size)
{
    min_count = 6;
    int size_1 = size / 3;
    int size_2 = 2 * size_1;
    int coordinate_1 = size_1 + 1;
    int coordinate_2 = size_2 + 1;
    solution[0] = {1, 1, size_2};
    solution[1] = {1, coordinate_2, size_1};
    solution[2] = {coordinate_2, 1, size_1};
    solution[3] = {coordinate_1, coordinate_2, size_1};
    solution[4] = {coordinate_2, coordinate_1, size_1};
    solution[5] = {coordinate_2, coordinate_2, size_1};
    return;
}

//Случай, когда сторона делится на 5
void formAnswer_multOf5(square_list &solution, int &min_count, int size)
{
    min_count = 8;
    int size_1 = size / 5;
    int size_2 = 2 * size_1;
    int size_3 = 3 * size_1;
    int coordinate_1 = size_2 + 1;
    int coordinate_2 = size_3 + 1;
    int coordinate_3 = 4 * size_1 + 1;
    solution[0] = {1, 1, size_3};
    solution[1] = {1, coordinate_2, size_2};
    solution[2] = {coordinate_2, 1, size_2};
    solution[3] = {coordinate_2, coordinate_2, size_2};
    solution[4] = {coordinate_2, coordinate_1, size_1};
    solution[5] = {coordinate_3, coordinate_1, size_1};
    solution[6] = {coordinate_1, coordinate_2, size_1};
    solution[7] = {coordinate_1, coordinate_3, size_1};
    return;
}

//Функция перебора вариантов
void backtracking(square_list &solution, int &min_count,
                  square_list &now_solution, int now_count,
                  int **field, int field_size)
{
    if (now_count == min_count)
        return;
    if (is_full(field, field_size) && now_count < min_count)
```

```
    {
        min_count = now_count;
        for (int i = 0; i < min_count; i++)
            solution[i] = now_solution[i];
    }
    int start_size = 2 * field_size / 3;
    for (int i = 0; i < field_size; i++)
        for (int j = 0; j < field_size; j++)
            if (field[i][j] == 0)
            {
                for (int size = start_size; size >= 1; size--)
                    if (is_enough_space(field, field_size, i, j, size))
                    {
                        Square new_elem = {i, j, size};
                        now_solution[now_count] = new_elem;
                        insert_square(field, new_elem);
                        backtracking(solution, min_count, now_solution,
                                     now_count + 1, field, field_size);
                        delete_square(field, new_elem);
                    }
                return;
            }
    return;
}

//Обёртка на функцией выполнения бектрекинга, выполняющая приближение к ответу
void backtracking_main(square_list &solution, int &min_count, int size)
{
    square_list now_solution = new Square[min_count];

    //Первое приближение
    int size_1 = size / 2;
    int size_2 = size_1 + 1;
    int coordinate_1 = size_2;
    int coordinate_2 = size_2 + 1;
    now_solution[0] = {coordinate_1, coordinate_1, size_2};
    now_solution[1] = {1, coordinate_2, size_1};
    now_solution[2] = {coordinate_2, 1, size_1};

    //Подготовка к бектрекингу
    int field_size = size_2;
    int **square_field = new int *[field_size];
    square_field[0] = new int[field_size * field_size];
    for (int i = 1; i < field_size; i++)
        square_field[i] = square_field[i - 1] + field_size;
    for (int i = 0; i < field_size; i++)
        for (int j = 0; j < field_size; j++)
        {
            if ((i == field_size - 1) && j == (field_size - 1))
                square_field[i][j] = 1;
            else
                square_field[i][j] = 0;
        }

    //Перебор вариантов
    backtracking(solution, min_count, now_solution, 3, square_field, field_size);

    //Корректировка координат обрезков решения
    for (int i = 3; i < min_count; i++)
    {
        solution[i].x += 1;
        solution[i].y += 1;
    }
```

```
    delete [] square_field[0];
    delete [] square_field;
    delete [] now_solution;
    return;
}

int main()
{
    int min_count = 20; // Для рассматриваемого случая (square_size <= 40)
оценка корректная
    square_list solution = new Square[min_count];
    int square_size;

    cin >> square_size;
    if (square_size % 2 == 0)
        formAnswer_multOf2(solution, min_count, square_size);
    else if (square_size % 3 == 0)
        formAnswer_multOf3(solution, min_count, square_size);
    else if (square_size % 5 == 0)
        formAnswer_multOf5(solution, min_count, square_size);
    else
        backtracking_main(solution, min_count, square_size);

    cout << min_count << endl;
    for (int i = 0; i < min_count; i++)
        cout << solution[i].x << " " << solution[i].y << " " << solution[i].size
<< endl;

    delete [] solution;
    return 0;
}
```