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

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Построение и анализ алгоритмов»

Тема: Поиск с возвратом

Студентка гр. 7304 _____ Нгуен Т.Т. Зуен

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2019

Цель работы:

Исследование алгоритмов поиска с возвратом, реализация программы заполнения квадрата минимальным количеством квадратов.

Задание:

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).

Экспериментальные результаты.

1. Был создан класс Square с полями Square(поле для заполнения квадратами), конструктур класса и метод void out() выхода результата.
2. start_square(int **square, int size, int &x, int &y) – проверка можно ли квадрат размера size установить, начиная с клетки с координатами (x,y).
3. fill_square(int **square, int x, int y, int w, int size) - проверка можно ли квадрат размера w существовать, начиная с клетки с координатами (x,y).

4. insert_square(int **square, int x, int y, int w, int var) - изменение каждую ячейку массива в квадрате с координатой верхнего левого угла (x, y) и размера w на var.
5. delete_square(int **square, int x, int y, int w) - удаления квадрата заданной стороны, верхний левый угол которого расположен в клетке с координатами (x,y).
6. side_square(int x, int y, int size) - возвращение длины квадрата.
7. backtracking(int **square,int size, int &minS, int count) – с помощью алгоритма поиска с возвратом разложить квадрат на минимальное меньших количество квадратов и их расположение.

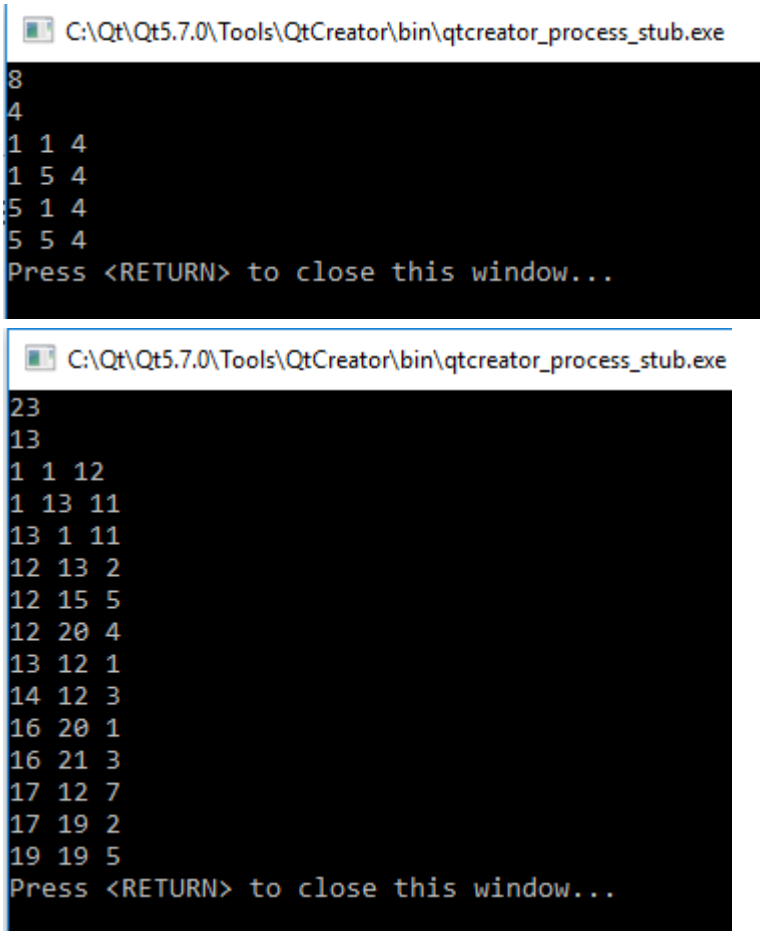
Код программы:

```
#include <iostream>
#include <vector>

using namespace std;

class Square{
public:
    int x, y, w;
    Square(int x, int y, int w) : x(x), y(y), w(w){}
    void out(int k){
        cout << x*k+1 << " " << y*k+1 << " " << w*k << endl;
    }
};

vector<Square> listSquare;
vector<Square> result;

bool start_square(int **square, int size, int &x, int &y){
    for(int i = 0; i < size; i++)
        for(int j = 0; j < size; j++)
            if(square[i][j] == 0){
                x = i;
                y = j;
                return true;
            }
    return false;
}

bool fill_square(int **square, int x, int y, int w, int size){
    if (x + w > size || y + w > size)
        return false;

    for(int i = x; i < x + w; i++)
        for(int j = y; j < y + w; j++)
            if(square[i][j] != 0)
                return false;
```

```
    return true;
}

void insert_square(int **square, int x, int y, int w, int var){
    for(int i = x; i < x + w; i++)
        for(int j = y; j < y + w; j++)
            square[i][j] = var;
}

void delete_square(int **square, int x, int y, int w){
    for(int i = x; i < x + w; i++)
        for(int j = y; j < y + w; j++)
            square[i][j] = 0;
}

int side_square(int x, int y, int size){
    if(size - x < size - y)
        return size - x;
    else
        return size - y;
}

void backtracking(int **square,int size, int &minS, int count){
    if(minS < count)
        return;

    int x_start = 0, y_start = 0;

    if(start_square(square, size, x_start, y_start) == false){
        if(count - 1 < minS){
            minS = count - 1;
            result = listSquare;
        }
    }
    else{
        int new_size = side_square(x_start, y_start, size);
        if(new_size > size -1){
            new_size = size - 1;
        }
        for(int i = new_size; i > 0; i--){
            if(fill_square(square, x_start, y_start, i, size)){
                insert_square(square, x_start, y_start, i, count);
                Square sq(x_start, y_start, i);
                listSquare.push_back(sq);
                backtracking(square, size, minS, count+1);
                listSquare.pop_back();
                delete_square(square, x_start, y_start, i);
            }
        }
    }
}

int main()
{
    int N;
    cin >> N;
```

```
int k = 0;
for(int i = 2; i <= N; i++){
    if(N % i == 0){
        k = N/i;
        N = i;
        break;
    }
}
 int minS = 20;

 int **square = new int*[N];
 for(int i = 0; i < N; i++){
     square[i] = new int[N];
     for(int j = 0; j < N; j++){
         square[i][j] = 0;
     }
 }

int size1 = (N + 1) / 2,
    size2 = N - (N + 1) / 2;

int count = 1;
insert_square(square, 0, 0, size1, count); count++;
insert_square(square, 0, size1, size2, count); count++;
insert_square(square, size1, 0, size2, count); count++;

listSquare.push_back(Square(0, 0, size1));
listSquare.push_back(Square(0, size1, size2));
listSquare.push_back(Square(size1, 0, size2));

 backtracking(square, N, minS, count);

 cout << minS << endl;
 for(int i = 0; i < result.size(); i++){
     result[i].out(k);
 }

 for (int i = 0; i < N; i++)
      delete[] square[i];
 delete[] square;

return 0;
 }
```

Результаты:



Выводы:

В результате работы программы исследовала алгоритм работы поиска с возвратом при помощи рекурсии, реализовала программы заполнения квадрата минимальным количеством квадратов.