

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

Есиков О.И.

Преподаватель

Филатов А.Ю.

Санкт-Петербург

2019

Цель работы.

Изучение алгоритма поиска с возвратом и его улучшений для ускорения перебора, реализация программы замощения квадрата наименьшим числом меньших квадратов.

Формулировка задания.

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).

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 ($2 \leq N \leq 20$).

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

Исходный код.

Была написана программа на языке c++, в которой реализован класс для работы с квадратом и функции «Even», «Odd3», «Odd5», которые обрабатывают частные случаи исходного размера квадрата.

```
#include <iostream>
#include <cmath>

using namespace std;

//структура для хранения информации о вставленных квадратах
struct Point
{
    int X;
    int Y;
```

```

    int space;
    Point()
    {
        X = 0;
        Y = 0;
        space = 0;
    }
};

//обработка чисел, кратных двум
void Even(int N)
{
    cout << "4" << endl;
    cout << "1 1 " << N/2 << endl << "1 " << N/2 + 1 << " " << N/2 << endl;
    cout << N/2 + 1 << " " << "1 " << N/2 << endl << N/2 + 1 << " " << N/2 + 1
<< " " << N/2 << endl;
}

//обработка чисел, кратных трём и не кратных двум
void Odd3(int N)
{
    cout << "6" << endl;
    cout << "1 1 " << 2*N/3 << endl;
    cout << "1 " << 2*N/3 + 1 << " " << N/3 << endl;
    cout << N/3 + 1 << " " << 2*N/3 + 1 << " " << N/3 << endl;
    cout << 2*N/3 + 1 << " " << 2*N/3 + 1 << " " << N/3 << endl;
    cout << 2*N/3 + 1 << " " << N/3 + 1 << " " << N/3 << endl;
    cout << 2*N/3 + 1 << " 1 " << N/3 << endl;
}

//обработка чисел, кратных 5, но не кратных трём
void Odd5(int N)
{
    cout << "8" << endl;
    cout << "1 1 " << 3*N/5 << endl;
    cout << "1 " << 3*N/5 + 1 << " " << 2*N/5 << endl;
    cout << 2*N/5 + 1 << " " << 3*N/5 + 1 << " " << 2*N/5 << endl;
    cout << 3*N/5 + 1 << " 1 " << 2*N/5 << endl;
    cout << 3*N/5 + 1 << " " << 2*N/5 + 1 << " " << N/5 << endl;
    cout << 4*N/5 + 1 << " " << 2*N/5 + 1 << " " << N/5 << endl;
    cout << 4*N/5 + 1 << " " << 3*N/5 + 1 << " " << N/5 << endl;
    cout << 4*N/5 + 1 << " " << 4*N/5 + 1 << " " << N/5 << endl;
}

class Square
{
private:
    Point* result; //информация о квадратах в лучшей конфигурации
    int** data;    //поле для работы
    int** best;    //лучшая достигнутая конфигурация
    int size;      //размер поля
    int min;       //число квадратов в лучшей конфигурации

public:
    Square(int N)
    {
        min = 9999; //очень большое, чтобы первый удачный вариант
        заменил
        size = N/2 + 1;

        data = new int*[size];
        for(int i(0); i < size; i++)
            data[i] = new int[size];
    }
};

```

```

        best = new int*[size];
        for(int i(0); i < size; i++)
            best[i] = new int[size];

        result = new Point[20]; //больше 20 не понадобится
    }

~Square()
{
    for(int i(0); i < size; i++)
        delete [] data[i];
    delete [] data;

    for(int i(0); i < size; i++)
        delete [] best[i];
    delete [] best;

    delete [] result;
}

void Init() //инициализация поля
{
    for(int i(0); i < size; i++)
    {
        for(int j(0); j < size; j++)
            data[i][j] = 0;
    }
    data[0][0] = -1; //часть большого квадрата, поэтому не используем
}

void Print() const //выводит текущую конфигурацию поля
{
    for(int i(0); i < size; i++)
    {
        for(int j(0); j < size; j++)
            cout << data[i][j] << " ";
        cout << endl;
    }
    cout << endl;
}

void Paint(int X, int Y, int N, int color) //рисует квадрат стороной N
цвета color
{
    for(int i(0); i < N; i++)
        for(int j(0); j < N; j++)
            data[X+i][Y+j] = color;
}

void Empty(int X, int Y, int N) //стирает квадрат стороной N
{
    for(int i(0); i < N; i++)
        for(int j(0); j < N; j++)
            data[X+i][Y+j] = 0;
}

bool isPaint(int X, int Y, int N) const //проверяет, можно ли нарисовать
квадрат стороной N
{
    if(X+N > size) //выход за пределы матрицы
        return false;
    if(Y+N > size)

```

```

        return false;

    for(int i(0); i < N; i++)    //проход по границе квадрата
    {
        if(data[X+i][Y] != 0)
            return false;
        if(data[X+i][Y+N-1] != 0)
            return false;
    }

    for(int j(0); j < N; j++)
    {
        if(data[X][Y+j] != 0)
            return false;
        if(data[X+N-1][Y+j] != 0)
            return false;
    }

    return true;
}

void Backtracking() //запуск перебора
{
    Paint(1, 0, 1, 1);    //всегда

    Paint(0, 1, 2, 2);    //первый вариант
    Step(0, 3, 4*size/5, 3);    //4*size/5 - теоретическое ограничение
    //максимального
    Empty(0, 1, 2);    //размера квадрата, который может быть внутри

    Paint(2, 0, 1, 2);    //второй вариант
    Paint(0, 1, 3, 3);
    Step(0, 4, 4*size/5, 4);
}

void Step(int X, int Y, int N, int color)    //перебор с возвратом
{
    if(!N)
        return;
    if(color > min)    //не рассматриваем варианты, если квадратов
        return;    //стало больше, чем в лучшем варианте
    if(Y >= size)    //переход на следующую строку
    {
        Y = 0;
        X++;
    }
    if(X >= size)    //прошли весь квадрат
    {
        if(color - 1 < min) //так как в функцию передалось color++
        {
            //запоминаем лучшую конфигурацию
            min = color - 1;
            for(int i(0); i < size; i++)
            {
                for(int j(0); j < size; j++)
                    best[i][j] = data[i][j];
            }
        }
        return;
    }

    while(N)    //перебор всех возможных квадратов
    {
        if(isPaint(X, Y, N))

```

```

        {
            Paint(X, Y, N, color);
            Step(X, Y+N, 4*size/5, color+1);
            Empty(X, Y, N);
        }
        N--;
    }

    if(data[X][Y]) //не оставляем дыры
        Step(X, Y+1, 4*size/5, color);
}

void Total() //вывод итоговых результатов
{
    int temp, length = 0;
    for(int i(0); i < size; i++)
    {
        for(int j(0); j < size; j++)
        {
            if(!i && !j) //не рассматриваем часть большого
                continue;
            temp = best[i][j];
            if(!result[temp].space) //цвет квадрата = его индекс в
                массиве
            {
                result[temp].X = i+1; //координаты квадрата
                result[temp].Y = j+1;
                length++; //сколько всего квадратов
            }
            result[temp].space++; //площадь квадрата
        }
    }

    cout << min + 3 << endl; //число квадратов + 3, расположение
    //которых знаем заранее
    cout << "1 1 " << size << endl; //эти 3 квадрата всегда расположены
    //здесь
    cout << "1 " << size + 1 << " " << size - 1 << endl;
    cout << size + 1 << " 1 " << size - 1 << endl;

    for(int i(1); i <= length; i++)
    {
        cout << result[i].X + size - 1 << " " << result[i].Y + size - 1 << "
"; //переход от квадрата N/2 + 1
        cout << (int)sqrt((double)result[i].space) << endl;
        //к N; из площади получаем сторону
    }
}

};

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

    //вызов обработчиков частных случаев
    if(!(N & 1))
    {
        Even(N);
        return 0;
    }
    if(!(N % 3))

```

```

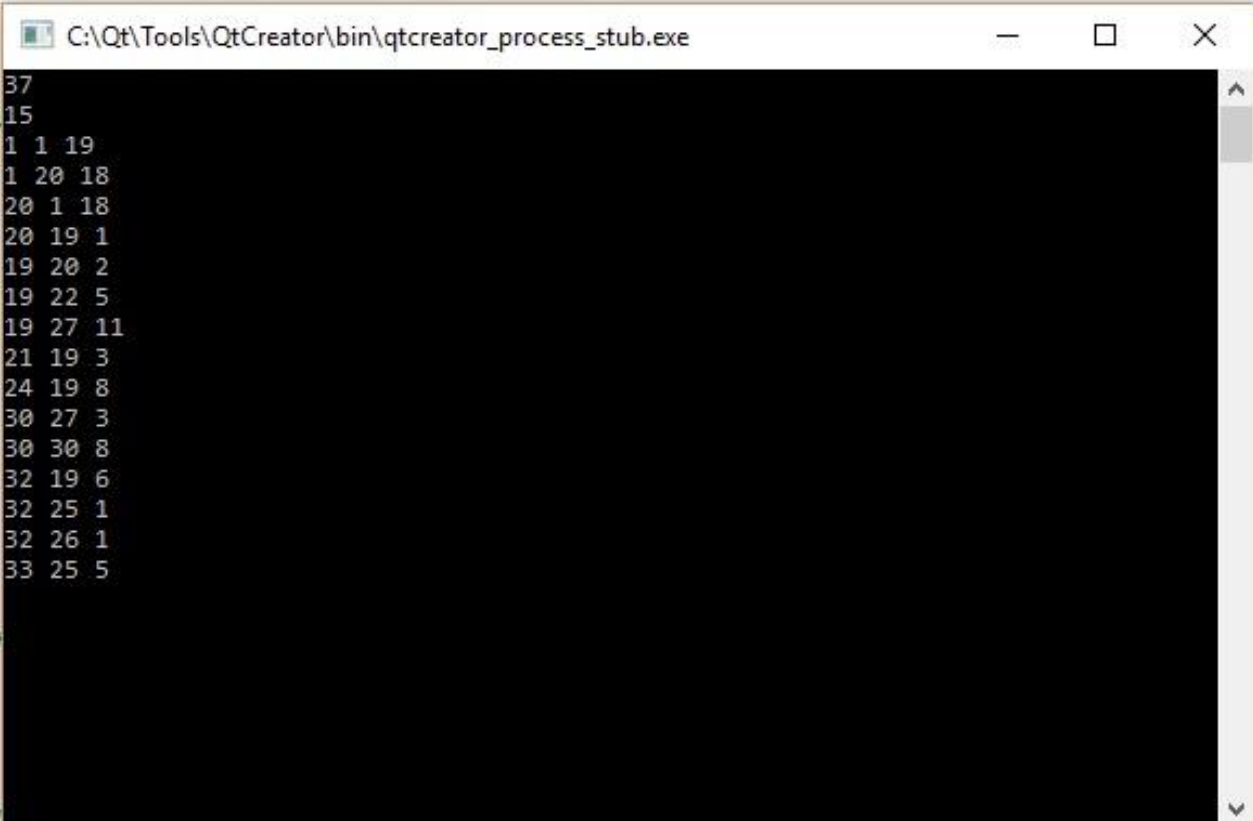
    {
        Odd3 (N) ;
        return 0;
    }
    if (! (N % 5))
    {
        Odd5 (N) ;
        return 0;
    }

    Square object (N) ;
    object.Init () ;
    object.Backtracking () ;
    object.Total () ;

    return 0;
}

```

Результат работы программы.



```

37
15
1 1 19
1 20 18
20 1 18
20 19 1
19 20 2
19 22 5
19 27 11
21 19 3
24 19 8
30 27 3
30 30 8
32 19 6
32 25 1
32 26 1
33 25 5

```

Выводы.

В ходе выполнения данной лабораторной работы был изучен алгоритм поиска с возвратом и его улучшения для ускорения перебора, была реализована программа замощения квадрата наименьшим числом квадратов, имеющих меньшую сторону.