

**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: Бэктрекинг**

Студент гр. 9382

\_\_\_\_\_

Демин В.В.

Преподаватель

\_\_\_\_\_

Фирсов М.А.

Санкт-Петербург

2020

### **Цель работы.**

Ознакомиться с бэктрекингом на примере решения прикладной задачи.

### **Задание.**

**Вариант 2и. Итеративный бэктрекинг. Исследование времени выполнения от размера квадрата.**

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  $7 \times 7$  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  $N$  ( $2 \leq N \leq 20$ ).

Выходные данные

Одно число  $KK$ , задающее минимальное количество обрезков(квадратов), из которых можно построить столешницу(квадрат) заданного размера  $NN$ . Далее должны идти  $KK$  строк, каждая из которых должна содержать три целых числа  $x, ux, u$  и  $ww$ , задающие координаты левого верхнего угла ( $1 \leq x, u \leq N$ ) 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

### **Алгоритм.**

Суть алгоритма заключается в том, чтобы перебрать все варианты расположения квадратов, чтобы найти минимальный. В частности, чтобы перебирать варианты, происходит следующие, в стеке хранится текущая расстановка квадратов. Начиная с вершины проверяем квадраты, пока не доходим до квадрата, который можно уменьшить на 1. Соответственно, так как квадрат ставится максимального размера, и постепенно будет уменьшаться, то так мы сможем перебрать все варианты.

### **Методы оптимизации.**

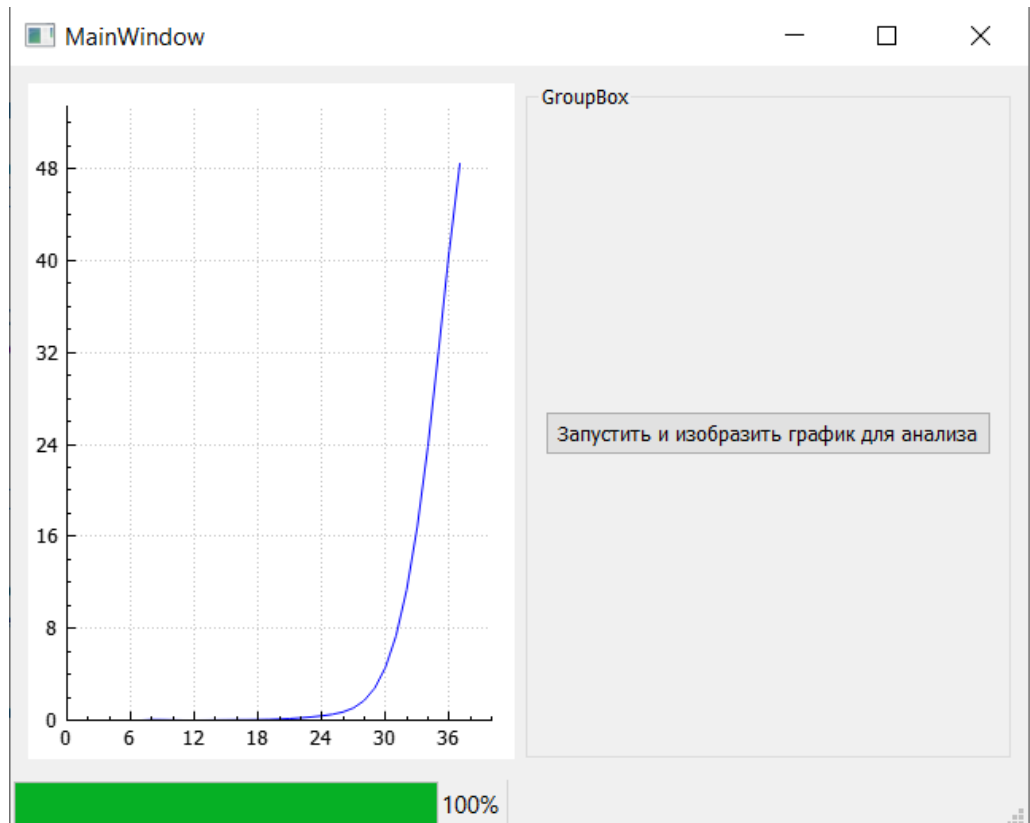
- Во-первых, присутствуют частные случаи для основного квадрата, размером кратным 2,3,5. Соответственно, кратным 2 будет соответственно разделенный на 4 квадрата, кратным 3 – 6 квадратов, кратным 5 – 8 квадратов.
- Во-вторых, заполнение квадратов не будет превышать текущего минимального количества заполнения квадрата.
- В-третьих, изначально сразу располагаются квадраты размером  $N/2+1$ ,  $N/2$ ,  $N/2$ , это сильно сокращает площадь квадрата на котором нужно расположить квадраты, также если один из квадратов будет меньше по размеру, то количество всех квадратов увеличиться в худшем случае на  $N/2$ .

### **Сложность алгоритма по операциям и по памяти**

Сложность алгоритма по памяти –  $O(N^2+k*N^2)=O(N^2)$

Где  $N$  – длина квадрата,  $0 \leq k \leq 1$  – коэффициент, который определяет сколько памяти хранит текущее минимальное расположение квадратов.

Первое слагаемое определяет определяет матрицу, на которой будут располагаться квадраты, второе – это текущие решение задачи, которое хранится в стеке. Максимальное возможное, это если расположены квадраты размером  $1 \times 1$ . Но с помощью методов оптимизации такое значение не будет получено.



Разработанная программа может строить график зависимости времени от длины квадрата. Для каждого значения было найдено решение, частные случаи были упущены в анализе, так как их время выполнения  $O(1)$ . Далее по полученным значениям методом Интерполяции были найдены промежуточные значения, чтобы определить общую форму графика. По результатам можно сделать вывод, что алгоритм имеет экспоненциальную сложность.

### **Функции и структуры данных.**

1. Структура `res` содержит в себе координаты  $x$ ,  $y$  и размер квадрата.
2. `Stack<res>` используется для хранения решения задачи. То есть содержит в себе квадраты, которые покрывают основную.
3. `class Square` – класс решения определенной задачи.

Поля класса `Square`

- `vector<vector<int>> data` – матрица на которой располагаются квадраты
- `int size` – размер матрицы
- `int colors` – id цвета квадрата на матрице

Методы класса `Square`

- `void makeSquare(int x, int y, int size)` –  $x, y$  – координаты квадрата, который нужно построить, `size` – размер этого квадрата
- `void clearSquare(int x, int y, int size)` -  $x, y$  – координаты квадрата, который нужно удалить, `size` – размер этого квадрата
- `int getSize(int x, int y)` -  $x, y$  – координаты в который нужно расположить квадрат с максимально возможным размером. Возвращает этот размер.
- `stack<res> make()` – основная функция выполнения задачи, возвращает стек с решением задачи.

### **Выводы.**

В данной задаче был изучен метод бектрекинга и возможные пути решения оптимизации в прикладной задаче.

### Тестирование.

№ п/п	Входные данные	Выходные данные	Комментарии
1	2	4 2 2 1 2 1 1 1 2 1 1 1 1	
2	3	6 3 3 1 2 3 1 1 3 1 3 2 1 3 1 1 1 1 2	
3	7	9 6 6 2 6 4 2 5 7 1 5 4 1 4 7 1 4 5 2 1 5 3 5 1 3 1 1 4	
4	9	6 7 7 3 4 7 3 1 7 3 7 4 3	

		7 1 3 1 1 6	
5	5	8 5 3 1 4 3 1 3 5 1 3 4 1 4 4 2 1 4 2 4 1 2 1 1 3	
6	11	11 9 9 3 9 6 3 8 11 1 8 10 1 8 6 1 7 6 1 6 10 2 6 7 3 1 7 5 7 1 5 1 1 6	
7	25	8 21 11 5 16 11 5 11 21 5 11 16 5 16 16 10	



		1 16 10 16 1 10 1 1 15	
--	--	------------------------------	--

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: square.h

```
#ifndef SQUARE_H
#define SQUARE_H
#include "Res.h"
#include <vector>
#include <stack>
#include <algorithm>
#include "widget.h"
#include "QDebug"

using namespace std;

class Square {
    vector<vector<int>> data; // Матрица основного квадрата
    int size; // размер матрицы
    int colors; // id квадратов
    Widget* widget; // виджет для отрисовки квадрата

    void checkSize() {
        if (data.size() != this->size) {
            qDebug() << "error size vector" << endl;
        }
        for (auto it = data.begin(); it < data.end(); ++it) {
            if (it->size() != this->size) {
                qDebug() << "error size vector" << endl;
            }
        }
    }

    // создание квадрата
    void makeSquare(int x, int y, int size) {
        colors++;
        for (int i = x; i < x + size; ++i) {
            for (int j = y; j < y + size; ++j) {
                data[i][j] = colors;
            }
        }
    }
}
```

```

}
//удаление квадрата
void clearSquare(int x, int y, int size) {
    colors--;
    for (int i = x; i < x + size; ++i) {
        for (int j = y; j < y + size; ++j) {
            data[i][j] = 0;
        }
    }
}

//получение максимально возможного размера
int getSize(int x, int y) {
    int max = size / 2 + 1;
    while (x + max > size || y + max > size) {
        max--;
    }
    bool flag = true;
    while (flag) {
        for (int i = 0; i < max; ++i) {
            for (int j = 0; j < max; ++j) {
                if (data[x + i][y + j] == 0) {
                    continue;
                } else {
                    flag = false;
                    break;
                }
            }
        }
        if (!flag) {
            max--;
            flag = true;
        } else {
            break;
        }
    }
    return max;
}

```

```

static bool compaire(res a, res b) {
    return a.size < b.size;
}

//отрисовка квадрата
void printQT(stack<res> stackOfRes){
    if(widget){
        widget->printSquare(stackOfRes,size);
    }
}

public:

Square(int N,Widget* a) {
    this->widget=a;
    this->size = N;
    data.resize(N);
    for (auto it = data.begin(); it < data.end(); ++it) {
        it->resize(N);
    }
    colors = 0;
    checkSize();
}

//функция решения задачи
stack<res> make() {
    stack<res> a; //минимальное решение
    stack<res> temp;//текущее решение

    if (this->size % 2 == 0) { //решение квадрата размера
кратным 2
        int k = size / 2;
        for (int i = 0; i < size; i = i + k) {
            for (int j = 0; j < size; j = j + k) {
                makeSquare(i, j, k);
                a.push(res(i, j, k));
            }
        }
    }
}

```

```

        } else if (this->size % 3 == 0) { //решение квадрата
размера кратным 3
            makeSquare(0, 0, size / 3 * 2);
            temp.push(res(0, 0, size / 3 * 2));
            printQT(temp);

            makeSquare(2 * size / 3, 0, size / 3);
            temp.push(res(2 * size / 3, 0, size / 3));
            printQT(temp);

            makeSquare(2 * size / 3, size / 3, size / 3);
            temp.push(res(2 * size / 3, size / 3, size / 3));
            printQT(temp);

            makeSquare(0, 2 * size / 3, size / 3);
            temp.push(res(0, 2 * size / 3, size / 3));
            printQT(temp);

            makeSquare(size / 3, 2 * size / 3, size / 3);
            temp.push(res(size / 3, 2 * size / 3, size / 3));
            printQT(temp);

            makeSquare(2 * size / 3, 2 * size / 3, size / 3);
            temp.push(res(2 * size / 3, 2 * size / 3, size / 3));
            printQT(temp);

            a = temp;

        } else if (this->size % 5 == 0) { //решение квадрата
размера кратным 5
            makeSquare(0, 0, size * 3 / 5);
            temp.push(res(0, 0, size * 3 / 5));
            printQT(temp);

            makeSquare(size * 3 / 5, 0, size * 2 / 5);
            temp.push(res(size * 3 / 5, 0, size * 2 / 5));
            printQT(temp);

```

```

makeSquare(0, size * 3 / 5, size * 2 / 5);
temp.push(res(0, size * 3 / 5, size * 2 / 5));
printQT(temp);

makeSquare(size * 3 / 5, size * 3 / 5, size * 2 / 5);
temp.push(res(size * 3 / 5, size * 3 / 5, size * 2 /
5));

printQT(temp);
for (int i = 0; i < size; ++i) {
    for (int j = 0; j < size; ++j) {
        if (data[i][j] == 0) {
            int k = getSize(i, j);
            makeSquare(i, j, k);
            temp.push(res(i, j, k));
            printQT(temp);
        }
    }
}

a = temp;
} else {
    int minSize = size * size;
    //первые три квадрата размера N/2+1, N/2, N/2
    int kSize = getSize(0, 0);
    makeSquare(0, 0, kSize);
    temp.push(res(0, 0, kSize));

    int k = getSize(kSize, 0);
    makeSquare(kSize, 0, k);
    temp.push(res(kSize, 0, k));

    k = getSize(0, kSize);
    makeSquare(0, kSize, k);
    temp.push(res(0, kSize, k));

    bool flag = true;
    while (flag) {
        //заполнение квадрата

```

```

        for (int i = 0; i < size; ++i) {
            for (int j = 0; j < size && temp.size() <
minSize; ++j) {
                if (data[i][j] == 0) {
                    k = getSize(i, j);
                    makeSquare(i, j, k);
                    temp.push(res(i, j, k));
                    printQT(temp);
                    if (temp.size() > minSize) {
                        break;
                    }
                }
            }
        }
        //проверка является ли текущее решение меньше
текущего минимального
        if (temp.size() < minSize) {
            minSize = temp.size();
            a = temp;
        }
        //удаление квадратов размером 1
        while (temp.top().size == 1) {
            clearSquare(temp.top().x,          temp.top().y,
temp.top().size);

            temp.pop();
        }
        //проверка является закончены ли все варианты
расположения квадратов, если нет, то уменьшаем последний квадрат размером
больше 1 на 1
        if (temp.size() > 3) {
            clearSquare(temp.top().x,          temp.top().y,
temp.top().size);

            makeSquare(temp.top().x,          temp.top().y,
temp.top().size - 1);

            res resTemp = res(temp.top().x, temp.top().y,
temp.top().size - 1);

```

```

        temp.pop();
        temp.push(resTemp);
    } else {
        flag = false;
    }
}

}

printQT(a);
return a;
}
};

```

```

#endif // SQUARE_H

```

**Название файла: widget.h**

```

#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QPainter>
#include <stack>
#include "Res.h"
using namespace std;
namespace Ui {
class Widget;
}

class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();
    void printSquare(std::stack<res> a,int size);
    QColor randColor(int index);
    void next();
    void setStepByStep(bool temp);

```



```

        void clearData();

protected:
    /* Определяем виртуальный метод родительского класса
     * для отрисовки содержимого виджета
     * */
    void paintEvent(QPaintEvent *event);

private:
    Ui::Widget *ui;
    vector<std::stack<res>> resTable;
    int sizeField;
    bool stepByStep;
};

#endif // WIDGET_H

```

**Название файла: res.h**

```

#ifndef RES_H
#define RES_H
#include <iostream>

struct res {
    int x;
    int y;
    int size;

    res(int x, int y, int size) {
        this->x = x;
        this->y = y;
        this->size = size;
    }
};

#endif // RES_H

```

### Название файла: menu.h

```
#ifndef MENU_H
#define MENU_H
#include "mainwindow.h"
#include "analizwindow.h"

#include <QWidget>

namespace Ui {
class Menu;
}

class Menu : public QWidget
{
    Q_OBJECT

public:
    explicit Menu(QWidget *parent = nullptr);
    ~Menu();

private slots:
    void on_pushButton_clicked();

    void on_pushButton_2_clicked();

private:
    Ui::Menu *ui;
    MainWindow* w;
    AnalizWindow* aw;
};

#endif // MENU_H
```

### Название файла: mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "widget.h"
```

```

#include "square.h"
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_pushButton_clicked();

    void on_pushButton_3_clicked();

    void on_pushButton_2_clicked();

private:
    Ui::MainWindow *ui;
    Widget* widget;
};
#endif // MAINWINDOW_H

```

### Название файла: analizwindow.h

```

#ifndef ANALIZWINDOW_H
#define ANALIZWINDOW_H

#include <QMainWindow>
#include "square.h"
#include <time.h>
#include <QProgressBar>
#include <QVector>

namespace Ui {
class AnalizWindow;

```

```

}

class AnalizWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit AnalizWindow(QWidget *parent = nullptr);
    ~AnalizWindow();

private slots:
    void on_pushButton_clicked();

private:
    double Lagrange(double x, QVector<double> xv, QVector<double>
yv);
    Ui::AnalizWindow *ui;
    QProgressBar* bar;
};

```

```
#endif // ANALIZWINDOW_H
```

**Название файла: analizwindow.cpp**

```

#include "analizwindow.h"
#include "ui_analizwindow.h"

#include <QLabel>

```

```

AnalizWindow::AnalizWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::AnalizWindow)
{
    ui->setupUi(this);
    bar= new QProgressBar(this);
    ui->statusbar->addWidget(bar);
    bar->setMinimum(2);
    bar->setMaximum(37);
}

```

```

    }

    AnalizWindow::~~AnalizWindow()
    {
        delete ui;
    }

    double    AnalizWindow::Lagrange(double    x,    QVector<double>    xv,
    QVector<double> yv) { //Lagrange polynomial
        int size = xv.size(); //Количество точек (для удобства)
        double sum = 0; //Значение функции
        for(int i = 0; i < size; i++){
            double mul = 1; //Произведение
            for(int j = 0; j < size; j++){
                if(i!=j) mul *= (x - xv[j])/(xv[i]-xv[j]);
            }
            sum += yv[i]*mul;
        }

        return sum;
    }

    void AnalizWindow::on_pushButton_clicked()
    {
        QVector<double> time;
        QVector<double> N;
        bar->reset();
        for (int i =2;i<=37 ;++i ) {
            Square* a =new Square(i,nullptr);
            clock_t start = clock();
            a->make();
            clock_t end = clock();
            double seconds = (double) (end - start) /CLOCKS_PER_SEC;
            qDebug()<<QString::number(seconds,'f',12);
            time.push_back(seconds);
            N.push_back(i);
            //ui->statusbar->showMessage(QString::number(i)+":
"+QString::number(seconds));

```

```

        bar->setValue(i);
        delete a;
    }

    QMutableVectorIterator<double> i(time);
    QMutableVectorIterator<double> j(N);

    while(i.hasNext() && j.hasNext()) {
        i.next();
        int n = j.next();

        if(n!=2&&(n%2==0||n%3==0||n%5==0)){
            i.remove();
            j.remove();
        }
    }

    QVector<double> result;
    QVector<double> x;
    for(double i = 2;i<=37;i=i+1){
        result.push_back( Lagrange(i,N,time));
        x.push_back(i);
    }

    ui->widget->addGraph();
    ui->widget->graph(0)->setData(x,result);
    ui->widget->yAxis->setRange(0, result[result.size()-1]+5);
    ui->widget->xAxis->setRange(0, 40);
    ui->widget->replot();
}

```

**Название файла: analizwindow.cpp**

```

#include <QApplication>
#include <menu.h>
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

```

```

        Menu w;
        w.show();
        return a.exec();
    }

```

### Название файла: mainwindow.cpp

```

#include "mainwindow.h"
#include "ui_mainwindow.h"

```

```

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    widget=new Widget();
    ui->verticalLayout_2->addWidget(widget);

}

```

```

MainWindow::~MainWindow()
{
    delete ui;
}

```

```

void MainWindow::on_pushButton_clicked()
{
    widget->clearData();
    widget->setStepByStep(false);
    Square *a = new Square(ui->spinBox->value(),widget);
    stack<res> resTable = a->make();
    ui->textEdit->setText(    QString::number(resTable.size())+"\n"
);

        for (; !resTable.empty(); ) {
            ui->textEdit->setText(ui->textEdit->toPlainText()+
QString::number(resTable.top().x+1)+"
"+QString::number(resTable.top().y+1)+"
"+QString::number(resTable.top().size)+"\n");
            resTable.pop();

```

```

        }
        delete a;
    }

void MainWindow::on_pushButton_3_clicked()
{
    widget->next();
}

void MainWindow::on_pushButton_2_clicked()
{
    widget->clearData();
    widget->next();
    widget->setStepByStep(true);
    Square *a = new Square(ui->spinBox_2->value(), widget);
    stack<res> resTable = a->make();
    ui->textEdit->setText(QString::number(resTable.size()) + "\n"
);
        for (; !resTable.empty(); ) {
            ui->textEdit->setText(ui->textEdit->toPlainText() +
QString::number(resTable.top().x+1) + "
"+QString::number(resTable.top().y+1) + "
"+QString::number(resTable.top().size) + "\n");
            resTable.pop();
        }
        delete a;
    }

```

### Название файла: menu.cpp

```

#include "menu.h"
#include "ui_menu.h"

Menu::Menu(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Menu)
{
    ui->setupUi(this);
    w=new MainWindow();
    aw=new AnalizWindow();

```



```

}

Menu::~Menu()
{
    delete ui;
    delete aw;
    delete w;
}

void Menu::on_pushButton_clicked()
{
    this->hide();
    w->show();
}

void Menu::on_pushButton_2_clicked()
{
    this->hide();
    aw->show();
}

```

**Название файла:** widget.cpp

```

#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    ui->setupUi(this);
    stepByStep=false;
}

Widget::~Widget()
{
    delete ui;
}

/* Метод, в котором происходит рисование

```

```

* */
void Widget::paintEvent(QPaintEvent *event)
{
    Q_UNUSED(event);
    QPainter painter(this); // Создаём объект отрисовщика
    // Устанавливаем кисть абриса
    painter.setPen(QPen(Qt::black, 1, Qt::SolidLine, Qt::FlatCap));

    /* Проверяем, какой из радиобаттонов выбран
    * */

    int w=450;
    int h=450;
    painter.setBrush(QBrush(Qt::white, Qt::SolidPattern));
    painter.drawRect(0,0,h,w);
    while(resTable.size() && !resTable[0].empty()){
        res temp = resTable[0].top();
        painter.setBrush(QBrush(randColor(resTable[0].size()),
Qt::SolidPattern));

        int size=(int)((double)temp.size/sizeField*h);
        int x=(int)((double)temp.x/sizeField*h);
        int y=(int)((double)temp.y/sizeField*h);
        painter.drawRect(x, y, size, size);
        resTable[0].pop();
    }
    if(resTable.size()){
        resTable.erase(resTable.begin());
    }
}

void Widget::printSquare(std::stack<res> a,int size){
    sizeField=size;
    resTable.push_back(a);
    if(!stepByStep){
        repaint();
    }
}

```

```

    QColor Widget::randColor(int index){
        std::vector< QColor> colors = {Qt::black, Qt::red, Qt::darkRed,
Qt::green,Qt::yellow,Qt::blue,Qt::darkMagenta,Qt::darkCyan,Qt::darkYellow
,Qt::black,Qt::magenta, Qt::darkGreen,  Qt::darkGray, Qt::lightGray};
        const ptrdiff_t idx = index%colors.size();
        return  QColor( colors[idx] );
    }

void Widget::next(){
    repaint();
}

void Widget::setStepByStep(bool temp){
    stepByStep=temp;
}

void Widget::clearData(){
    resTable.clear();
}

```