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
Вариант 1р

Студент гр. 8383 _____ Шишкин И.В.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

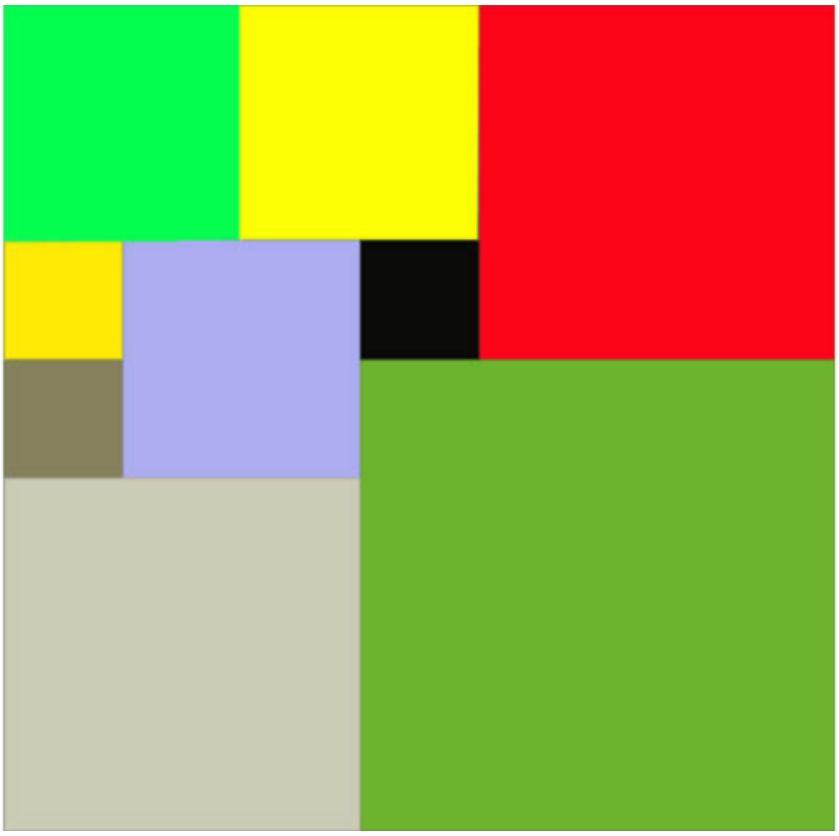
Цель работы.

Ознакомиться с методом поиска с возвратом и применить его на практике, решив задачу о заполнении квадрата.

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 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.

Например, столешница размера 7×7 может быть построена из 9 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 ($1 \le x, y \le N$) 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

Вар. 1р. Рекурсивный бэктрекинг. Выполнение на Stepik всех трёх заданий в разделе 2.

Описание алгоритма.

На вход подается размер квадрата N, который нужно составить из квадратов размером N-1. Всего есть 4 случая:

1) Число N - четное. Тогда квадрат разбивается на 4 квадрата, как показано на рис. 1.

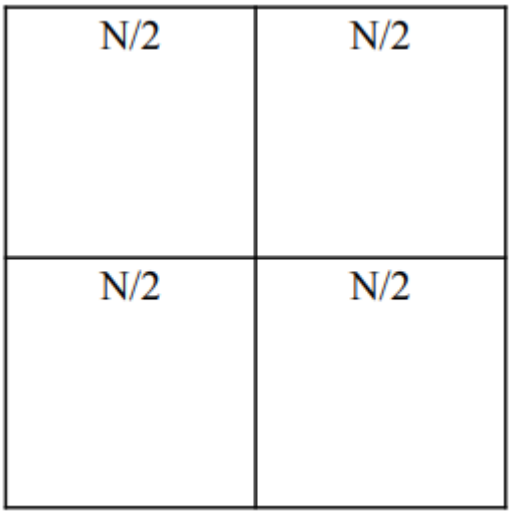


Рисунок 1 – Минимальное разбиение квадрата с четной стороной N

2) Число N делится на 3. Тогда квадрат разбивается на 6 квадратов, как показано на рис. 2.

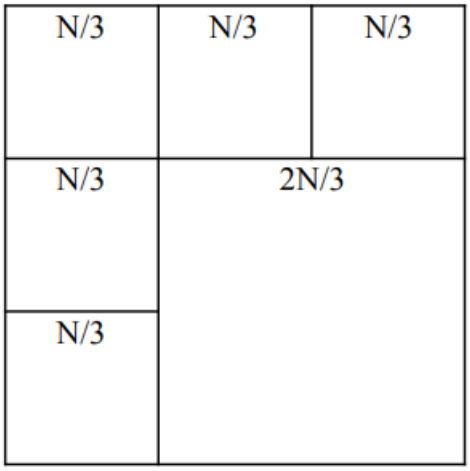


Рисунок 2 – Минимальное разбиение квадрата для N кратного 3

3) Число Nделится на 5. Тогда квадрат разбивается на 8 квадратов, как показано на рис. 3.

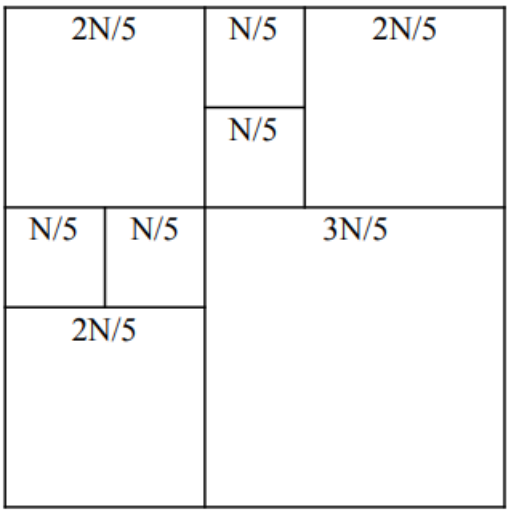


Рисунок 3 – Минимальное разбиение квадрата для N кратного 5

4) Число N- простое. В данном случае нет никаких очевидных ответов, поэтому именно для этого случая применяется рекурсивный бэктрекинг. При этом во всех вариантах, когда N - простое число, прослеживается закономерность. Первые 3 квадрата всегда ставятся так, как показано на рис. 4.

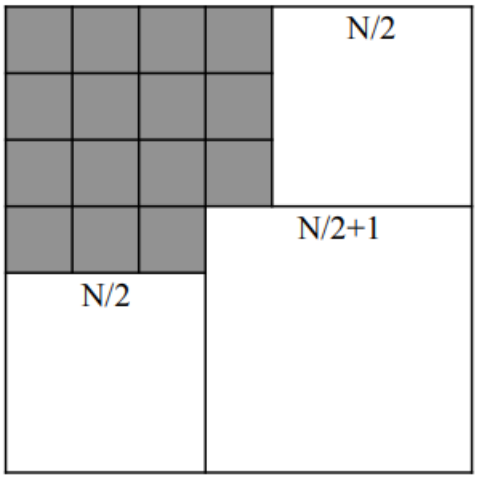


Рисунок 4 – Расположение квадратов

Далее в левый верхний угол ставится квадрат, размер которого изменяется от N/2 до 1. Затем, функция пытается вставить в столешницу квадраты разных размеров (от большего к меньшему). Например, если текущий размер квадрата в функции равен 3, то функция пытается найти с помощью другой функции check пустую область, куда можно такие квадраты вставить. При этом, после попытки вставки каждого размера квадрата, функция проверяет, минимальное ли это число квадратов. Реализована оптимизация для

сокращения времени выполнения программы: она проверяет сколько минимально еще может поместиться квадратов в столешницу, и, если это число в совокупности с числом квадратов, которые уже есть, оказывается больше минимального значения, то функция останавливает свое выполнение. После того, как были поставлены 3 квадрата, ставится квадрат в левый верхний угол, после этого ставится 5-й квадрат между 4 и верхним квадратом в правом углу, при этом размер 5-го квадрата будет не больше N/2+1-i.

Описание способов хранения частичных решений.

```
struct info {
      int x;
      int y;
      intsize;

      info()
};
```

Структура, созданная для удобного хранения координат квадратов и их размеров.

```
info coordRes[70];
```

coordRes - массив, в котором хранятся результирующие координаты и размеры квадратов.

```
int** a = new int* [N];
int** res = new int* [N];
```

Массивы для хранения промежуточного и минимального разбиения квадрата. Каждый квадрат имеет свой номер, этим номером и закрашиваются все ячейки, на которых располагается квадрат. Квадрат размером 1x1 отображается в массиве 0. На рис. 5 приведен пример отображения квадрата в массивах a и res. Цифрами 1 заполнен квадрат 4x4, цифрами 2 - квадрат 3x3, цифрами 3 - квадрат 3x3, цифрами 4, 5 и 6 - квадраты 2x2. Цифрами 0 - квадраты 1x1. Так как функция пытается вставить квадраты в столешницу от большего к меньшему, то тот факт, что единичные квадраты заполняются нулями, никак не влияет на правильность ответа.

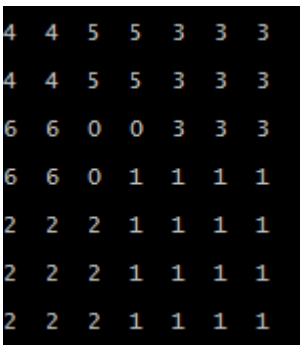


Рисунок 5 – Расположение квадратов в массивах

Описание функции drawingaSquare.

Функция `inline void drawingaSquare(int x, int y, int cycle, int counter, int** a)` рисует квадрат размером cycle в точке (x;y). Для удобства, соответствующие значения массива a заполняет значениями переменной counter.

Описание функции check.

Функция bool check(int** a, int& N, int& cycle, int& x, int& y) проверяет, можно ли в точку (x;y) поставить квадрат размером cycle. На вход подается двумерный массив a - столешница; N - размер столешницы.

Описание рекурсивной функции backTr.

Функция `void backTr(int N, int counter, int** a, int** res, int cycle, int area, info* coordRes, info* tmp)` - на вход подается размер столешницы N; счетчик квадратов внутри столешницы counter; двумерный массив для бэктрекинга a; результирующий двумерный массив res, хранящий минимальное расположение квадратов внутри столешницы; переменная, использованная для построения квадратов cycle; не занятая квадратами площадь area; coordRes – переменная, хранящая в себе результирующие данные о координатах квадратов и их размеров; tmp – промежуточные данные координат и размеров.

Затем идет проверка на то, квадрат какого размера функция пытается поставить. Если этот размер не равен 1, то с помощью функции checkпроизводится проверка: можно ли вписать квадрат такого размера в столешницу. Если вписать не получается, то функция backTr вызывается рекурсивно, но размер квадрата уменьшается на 1. В ином случае запускается цикл, изменяющий размер квадрата от имеющегося до 1. В начале цикла, так как квадрат получилось вписать, квадрат рисуется на столешнице с помощью функции drawingaSquare. Затем, функция вызывается рекурсивно с соответствующим уменьшением размера квадрата. Если же размер квадрата, который функция пытается поставить равен 1, значит остались только единичные квадраты, которые ищутся в цикле, а координаты записываются в переменные. Идет проверка, минимальное ли число квадратов получилось. Если да, то данные записываются в результирующие переменные.

Если в оставшуюся область можно вписать только квадраты размером 1, то функция подсчитывает оставшуюся площадь и никак не заполняет эти квадраты в двумерном массиве, следовательно, эти квадраты будут отображаться нулями.

Сложность алгоритма.

Сложность алгоритма по времени можно оценить, как

$$O\left(\left(\frac{N}{2}\right)^{\left(\frac{N}{2}\right)-1}\right).$$

Так как в худшем случае рекурсивная функция вызывается не более $\left(\frac{N}{2}-1\right)$ раза, а цикл в этой функции не более $\left(\frac{N}{2}\right)$ раза.

Сложность по памяти в худшем случае можно оценить, как

$$O(N^2).$$

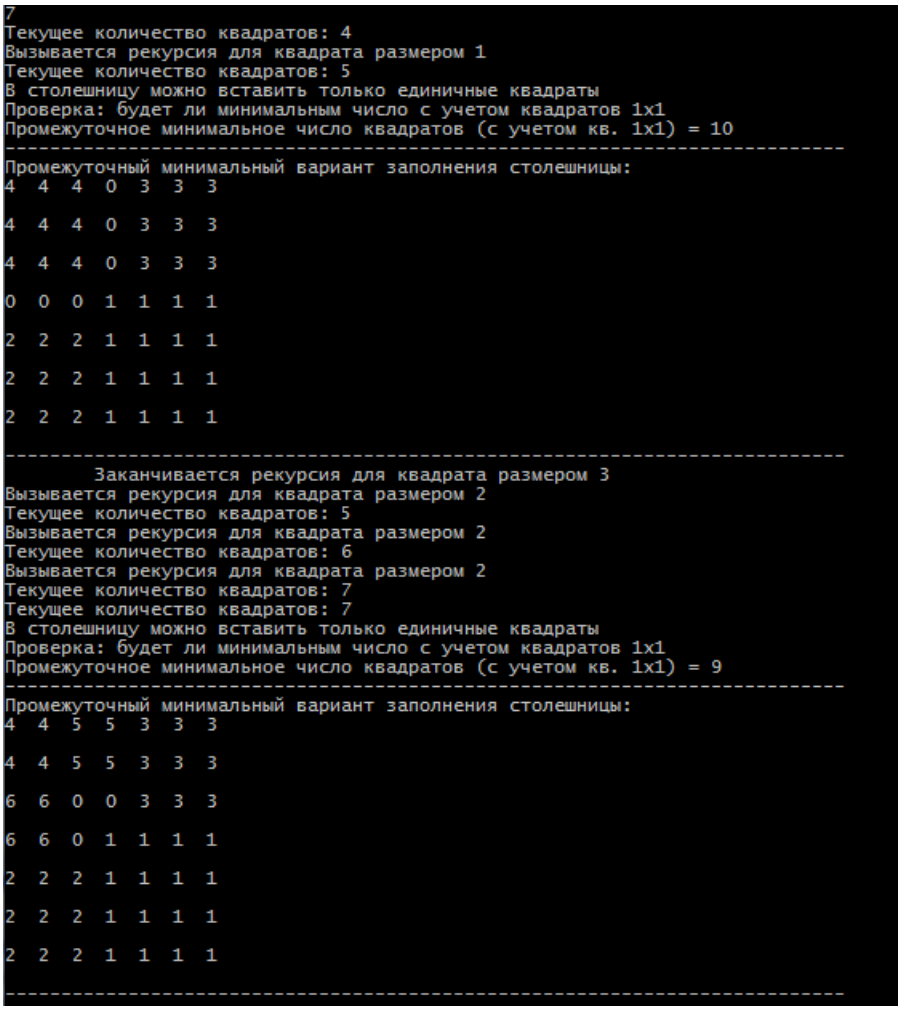
Так как хранится 2 двумерных массива.

Спецификация программы.

Программа написана на языке C++. Входные данные подаются из терминала. Результат программы выводится на экран. Пользователю нужно ввести лишь размер столешницы (N).

Тестирование.

Пример вывода результата для N = 7.



```
--------------------------------------------------------------------------------
        Заканчивается рекурсия для квадрата размером 2
Вызывается рекурсия для квадрата размером 2
Минимальный расчет квадратов, которые еще можно поставить в столешницу показал, что
это значение будет больше или равно минимальному, следовательно рассчет прекращается

        Заканчивается рекурсия для квадрата размером 1
        Заканчивается рекурсия для квадрата размером 2
Вызывается рекурсия для квадрата размером 2
Минимальный расчет квадратов, которые еще можно поставить в столешницу показал, что
это значение будет больше или равно минимальному, следовательно рассчет прекращается

        Заканчивается рекурсия для квадрата размером 1
        Заканчивается рекурсия для квадрата размером 2
Вызывается рекурсия для квадрата размером 3
Текущее количество квадратов: 5
Вызывается рекурсия для квадрата размером 3
Текущее количество квадратов: 6
Текущее количество квадратов: 6
Минимальный расчет квадратов, которые еще можно поставить в столешницу показал, что
это значение будет больше или равно минимальному, следовательно рассчет прекращается

        Заканчивается рекурсия для квадрата размером 3
Вызывается рекурсия для квадрата размером 3
Текущее количество квадратов: 6
Минимальный расчет квадратов, которые еще можно поставить в столешницу показал, что
это значение будет больше или равно минимальному, следовательно рассчет прекращается

        Заканчивается рекурсия для квадрата размером 2
Вызывается рекурсия для квадрата размером 3
Минимальный расчет квадратов, которые еще можно поставить в столешницу показал, что
это значение будет больше или равно минимальному, следовательно рассчет прекращается

        Заканчивается рекурсия для квадрата размером 1
        Заканчивается рекурсия для квадрата размером 1
--------------------------------------------------------------------------------
Ответ:
9
4 4 4
1 5 3
5 1 3
1 1 2
3 1 2
1 3 2
3 3 1
4 3 1
3 4 1

Время выполнения в сек.: 0
--------------------------------------------------------------------------------
Итоговое заполнение столешницы квадратами:

4  4  5  5  3  3  3

4  4  5  5  3  3  3

6  6  0  0  3  3  3

6  6  0  1  1  1  1

2  2  2  1  1  1  1

2  2  2  1  1  1  1

2  2  2  1  1  1  1
```

Input	Output
13	11 7 7 7 1 8 6 8 1 6 1 1 3 4 1 4 1 4 2 3 5 3

	6 5 2 1 6 2 3 4 1 6 7 1
19	13 10 10 10 1 11 9 11 1 9 1 1 6 7 1 4 7 5 4 1 7 4 5 7 2 5 9 2 7 9 2 9 9 1 10 9 1
2	4 1 1 1 2 1 1 1 2 1 2 2 1
37	15 19 19 19 1 20 18 20 1 18 1 1 12 13 1 7 13 8 7

	1 13 7 8 13 3 11 15 1 12 15 5 8 16 4 17 15 3 11 13 2 17 18 2 19 18 1

Вывод.

В ходе выполнения лабораторной работы был реализован алгоритм нахождения минимального разбиения квадрата на меньшие методом бэктрекинга.

ПРИЛОЖЕНИЕ

КОД ПРОГРАММЫ

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

usingnamespace std;

structinfo {
	int x;
	int y;
	int size;

	info() {
		x = 0;
		y = 0;
		size = 0;
	}
};

int minn = 1601;

inlinevoid drawingaSquare(intx, inty, intcycle, intcounter, int** a);
//рисуетквадрат
void backTr(intN, intcounter, int** a, int** res, intcycle, intarea, info* coordRes, info*
tmp);	//основнаярекурсивнаяфункция
bool check(int** a, int&N, int&cycle, int&x, int&y);	//проверяет, можноливкоординаты x
и y вписатьквадратразмером cycle

int main() {
	setlocale(0, "");
	int N;			//размерстолешницы
	info coordRes[70];	//результат координат и размеров
	info tmp[70];
	int K = 1;		//счетчик количества квадратов

	cin >> N;
	if (cin.fail()) {
		system("pause");
		return 0;
	}
	while (N< 2 || N> 40) {		//проверканато, что 2 <= N<= 40
		cout<<"Ндолжнобыть 2 <= N<= 40!\n";
		cin >> N;
	}

	//столешница
	int** a = newint* [N];
	int** res = newint* [N];
	for (int i = 0; i < N; i++) {
		a[i] = newint[N];
		res[i] = newint[N];

		for (int j = 0; j < N; j++) {	//обнуление
			a[i][j] = 0;
			res[i][j] = 0;
		}
	}

	if (N % 2 == 0) {		//если размер столешницы четный
		cout <<"4\n";
		int half = N / 2;
```

```
		cout <<"1 1 "<< half << endl;
		cout << half + 1 <<" 1 "<< half << endl;
		cout <<"1 "<< half + 1 <<" "<< half << endl;
		cout << half + 1 <<" "<< half + 1 <<" "<< half << endl;
	}

	elseif (N % 3 == 0) {		//если размер столешницы кратен 3
		cout <<"6\n";
		int bigSq = (N * 2) / 3;
		int smallSq = N - bigSq;

		cout <<"1 1 "<< bigSq << endl;
		cout << bigSq + 1 <<" 1 "<< smallSq << endl;
		cout << bigSq + 1 <<" "<< smallSq + 1 <<" "<< smallSq << endl;
		cout <<"1 "<< bigSq + 1 <<" "<< smallSq << endl;
		cout << smallSq + 1 <<" "<< bigSq + 1 <<" "<< smallSq << endl;
		cout << bigSq + 1 <<" "<< 2 * smallSq + 1 <<" "<< smallSq << endl;
	}

	elseif (N % 5 == 0) {		//если размер столешницы кратен 5
		int a = N / 5;

		cout <<"8"<< endl;
		cout <<"1 1 "<< 2 * a << endl;
		cout << 2 * a + 1 <<" 1 "<< a << endl;
		cout << 3 * a + 1 <<" 1 "<< 2 * a << endl;
		cout << 2 * a + 1 <<" "<< a + 1 <<" "<< a << endl;
		cout <<"1 "<< 2 * a + 1 <<" "<< a << endl;
		cout << a + 1 <<" "<< 2 * a + 1 <<" "<< a << endl;
		cout << 2 * a + 1 <<" "<< 2 * a + 1 <<" "<< 3 * a << endl;
		cout <<"1 "<< 3 * a + 1 <<" "<< 2 * a << endl;
	}

	else {					//если размер столешницы - простое число
		longlong start = 0;
		longlong end = 0;
		start = clock();
		int area = pow(N / 2 + 1, 2) - 1;
		drawingaSquare(N / 2, N / 2, N / 2 + 1, K, a);
		tmp[1].x = N / 2;
		tmp[1].y = N / 2;
		tmp[1].size = N / 2 + 1;
		K++;

		drawingaSquare(N / 2 + 1, 0, N / 2, K, a);
		tmp[2].x = N / 2 + 1;
		tmp[2].y = 0;
		tmp[2].size = N / 2;
		K++;

		drawingaSquare(0, N / 2 + 1, N / 2, K, a);
		tmp[3].x = 0;
		tmp[3].y = N / 2 + 1;
		tmp[3].size = N / 2;
		K++;

		backTr(N, K, a, res, N / 2, area, coordRes, tmp);
		cout << endl << minn << endl;
		for (int i = 1; i < minn + 1; i++) {
			cout << coordRes[i].y + 1 <<" "<< coordRes[i].x + 1 <<" "<<
coordRes[i].size << endl;
		}
		end = clock();
```

```
		cout << endl <<"Времявыполненияв сек.: "<< (end - start) / CLOCKS_PER_SEC<<
endl;
		cout << endl;

		cout <<"Result:\n\n";
		for (int i = 0; i < N; i++) {
			for (int j = 0; j < N; j++) {
				if (res[i][j] >= 10) cout << res[i][j] <<" ";
				else cout << res[i][j] <<"  ";
			}
			cout << endl << endl;
		}
	}

	for (int i = 0; i < N; i++) {
		delete[] a[i];
		delete[] res[i];
	}
	delete[] a;
	delete[] res;
	return 0;
}

inlinevoid drawingaSquare(intx, inty, intcycle, intcounter, int** a) {
	for (int i = x; i <x + cycle; i++) {
		for (int j = y; j <y + cycle; j++) {
			a[i][j] = counter;
		}
	}
}

void backTr(intN, intcounter, int** a, int** res, intcycle, intarea, info * coordRes, info *
tmp) {
	if (cycle<= 0) return;

	int areaTmp = area - 1;
	int counterTmp = counter;

	for (int i = cycle; i > 0; i--) {	//оптимизация: примерный расчет квадратов,
которые еще можно поставить
		if (i * i <= areaTmp) {
			int k = areaTmp / (i * i);
			counterTmp += k;
			areaTmp -= k * i * i;
		}
	}

	if (minn <= areaTmp + counterTmp) return;

	if (cycle != 1) {		//если размер проверяемого квадрата не равен 1
		int x = 0;
		int y = 0;
		if (check(a, N, cycle, x, y) &&cycle * cycle<area) {
			for (int i = cycle; i > 0; i--) {
				drawingaSquare(x, y, i, counter, a);

				tmp[counter].x = x;
				tmp[counter].y = y;
				tmp[counter].size = i;

				if (counter == 4) {
					//cout << "Вызывается рекурсия для квадрата размером " <<
N / 2 + 1 - i << endl;
```

```
                                        backTr(N, counter + 1, a, res, N / 2 + 1 - i, area - i *
i, coordRes, tmp);
                                }
                                else {
                                        //cout << "Вызывается рекурсия для квадрата размером " <<
cycle << endl;
                                        backTr(N, counter + 1, a, res, cycle, area - i * i,
coordRes, tmp);
                                }

                                tmp[counter].x = 0;       //обнуление
                                tmp[counter].y = 0;
                                tmp[counter].size = 0;

                                drawingaSquare(x, y, i, 0, a);
                                //cout << "Заканчивается рекурсия для квадрата размером " << i <<
endl;
                        }
                }
                else backTr(N, counter, a, res, cycle - 1, area, coordRes, tmp);
        }
        else {
                if (minn >counter + area - 1) {
                        for (int i = 0; i <N / 2 + 1; i++) {
                                for (int j = 0; j <N / 2 + 1; j++) {
                                        if (a[i][j] == 0) {
                                                counter++;
                                                tmp[counter - 1].x = i;
                                                tmp[counter - 1].y = j;
                                                tmp[counter - 1].size = 1;
                                        }
                                }
                        }
                        counter--;
                        minn = counter;

                        for (int i = 1; i <counter + 1; i++) {
                                coordRes[i].x = tmp[i].x;
                                coordRes[i].y = tmp[i].y;
                                coordRes[i].size = tmp[i].size;
                        }
                        for (int i = 0; i <N; i++)
                                for (int j = 0; j <N; j++)
                                        res[i][j] = a[i][j];

                        cout <<"Промежуточный минимальный вариант заполнения столешницы:"<<
endl;
                        for (int i = 0; i <N; i++) {
                                for (int j = 0; j <N; j++) {
                                        if (res[i][j] >= 10) cout <<res[i][j] <<" ";
                                        else cout <<res[i][j] <<"  ";
                                }
                                cout << endl << endl;
                        }
                        cout << endl << endl;
                }
        }
}

bool check(int** a, int&N, int&cycle, int&x, int&y) {
        bool checker = true;

        for (int i = 0; i <N - (N / 2) - cycle + 1; i++) {
                for (int j = 0; j <N - (N / 2) - cycle + 1; j++) {
```

```
                    if (a[i][j] == 0) {          //если нашелся пустой элемент, то
проверяем, можно ли в это место вписать квадрат размером cycle
                            checker = true;
                            for (int k = 0; k <cycle; k++) {
                                    if (a[i + k][j] || a[i][j + k] || a[i + cycle - 1 - k][j +
cycle - 1] || a[i + cycle - 1][j + cycle - 1 - k]) {
                                            checker = false;
                                            break;
                                    }
                            }

                            if (checker) {
                                    y = j;
                                    x = i;
                                    returntrue;
                            }
                    }
            }
      }
      returnfalse;
}
```