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:

Студентка гр. 8383 _____ Максимова А.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

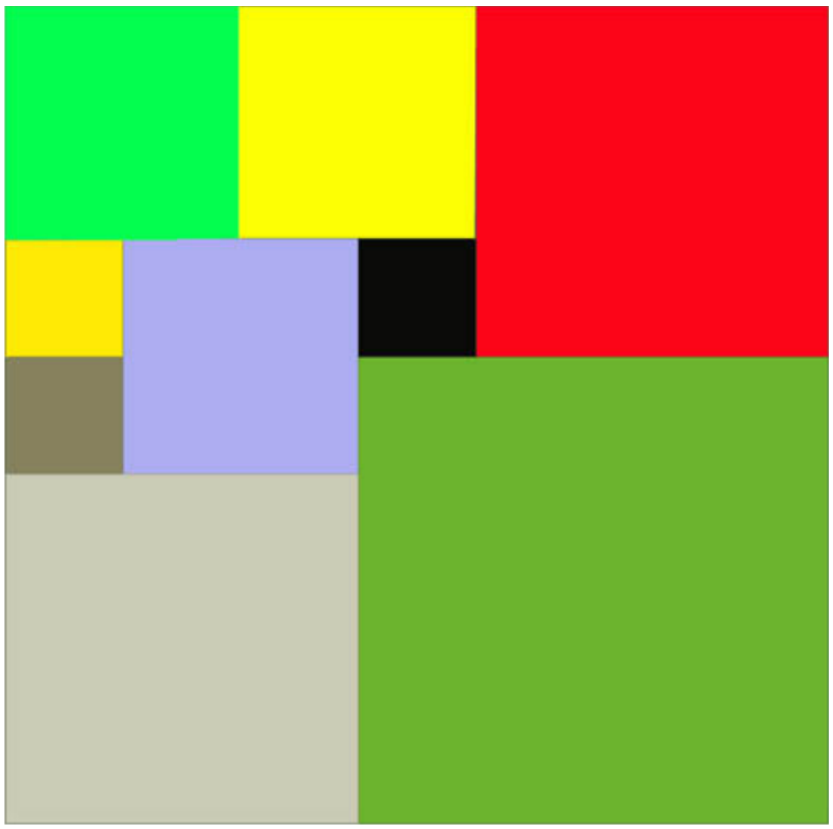
Цель работы.

Изучить алгоритм бэктрекинга, реализовать и оптимизировать его для решения поставленной задачи.

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).

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≤N≤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≤x, y≤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. 1и. Итеративный бэктрекинг. Поиск решения за разумное время (меньше минуты) для 2≤N≤30.

Основные теоретические положения.

Поиск с возвратом (бэктрекинг) - общий метод нахождения решений задачи, в которой требуется полный перебор всех возможных вариантов в некотором множестве M.

Решение задачи методом поиска с возвратом сводится к последовательному расширению частичного решения. Если на очередном шаге такое расширение провести не удается, то возвращаются к более короткому частичному решению и продолжают поиск дальше. Данный алгоритм позволяет найти все решения поставленной задачи, если они существуют. Для ускорения метода стараются вычисления организовать таким образом, чтобы как можно раньше выявлять заведомо неподходящие варианты. Зачастую это позволяет значительно уменьшить время нахождения решения.

Метод поиска с возвратом является универсальным. Достаточно легко проектировать и программировать алгоритмы решения задач с использованием этого метода. Однако время нахождения решения может быть очень велико даже при небольших размерностях задачи (количестве исходных данных), причём настолько велико (может составлять годы или даже века), что о практическом применении не может быть и речи. Поэтому при проектировании таких алгоритмов, обязательно нужно теоретически оценивать время их работы на конкретных данных.

Описание алгоритма.

В данной работе необходимо по введенному положительному числу sizeTable, интерпретируемому как размер стола, найти его оптимальное замощение квадратами, то есть используя их минимальное количество. При этом решение задачи должно основываться на использовании итеративного бэктрекинга. Таким образом, основная идея реализации алгоритма - это перебор всех возможных вариантов замощения с выбором оптимального.

Также решение задачи должно быть найдено за разумное время, что невозможно осуществить без оптимизации предложенного алгоритма.

Оптимизация для заведомо известных решений:

Пусть G(N) - минимальное число квадратов для замощения стола со стороной N. Тогда:

1) Оценка снизу G(N) = 4 - для любого N и абсолютно всегда четыре для любого четного числа, в таком случае сторона каждого квадрата равна N/2. Поэтому обработка четных N будет рассматриваться как частный случай, соответствующее решение представлено на рис. 1, где N = 8.

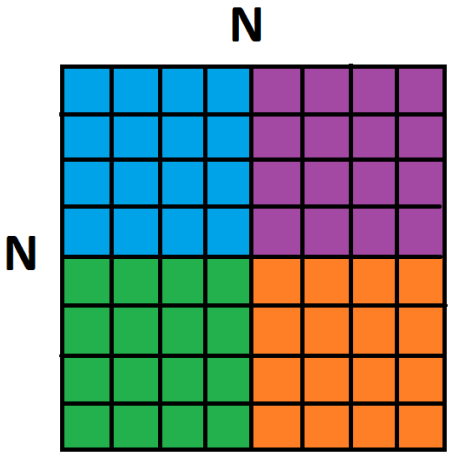


Рисунок 1 - Оптимальное разбиение для квадрата с четной N

2) Вторым частным случаем являются значения N кратные трём, минимальное разбиение при этом равно шести и состоит из одного квадрата со стороной $\frac{2}{3}$N и пяти квадратов со стороной $\frac{1}{3}$N. Соответствующее решение показано на рис. 2, где N = 9.

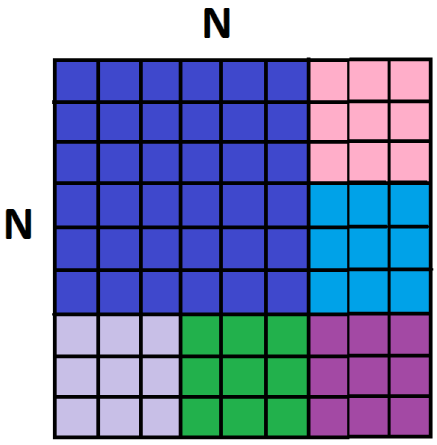


Рисунок 2 - Оптимальное разбиение для квадрата со стороной кратной трём

3) Третий частный случай - N кратно 5. Получаем восемь квадратов: наибольший со стороной $\frac{3}{5}$N, три квадрата - $\frac{2}{5}$N и четыре - $\frac{N}{5}$. Соответствующее разбиение представлено на рисунке 3, где N = 5.

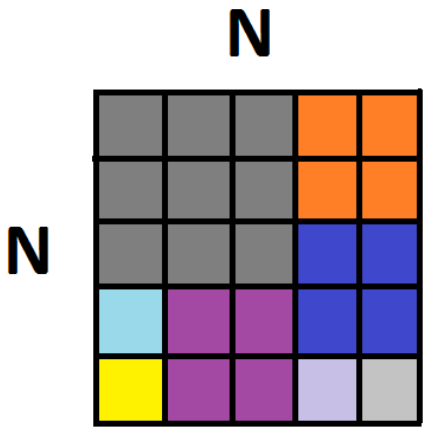


Рисунок 3 - Оптимальное разбиение для квадрата со стороной кратной пяти

4) Если N - составное число, то есть N = p1*p2*...*pk, то минимальное разбиение для такого квадрата, есть разбиение для N = pj = min(p1,...,pk). Ссоответствующее решение представлено на рис. 4, где N = 6.

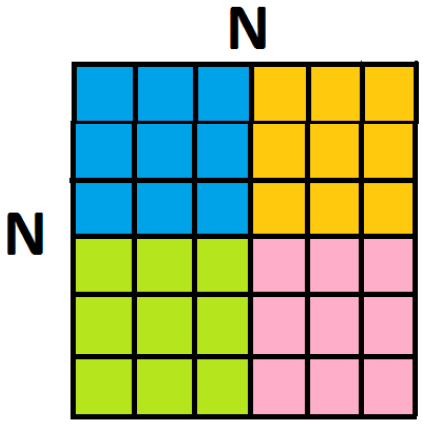


Рисунок 4 - Оптимальное разбиение для квадрата со стороной N - составным числом

5) Для простого N вида N=2k+1 получить заведомо известное решение не представляется возможным, поэтому для его нахождения нужно применить бэктрекинг, также с использованием оптимизаций, описанных ниже.

Первая оптимизация:

В любом квадрате со стороной N=2k+1 возможно построить три первых квадрата, используя приведенный ниже алгоритм:

1. Заполняем в одном из углов квадрат со стороной k+1 (например, правый нижний угол)
2. В соседнем углу строим максимально большой квадрат, он будет со стороной k (правый верхний угол)
3. Теперь рядом с квадратом из п.1 строим с другой стороны квадрат со стороной k (левый нижний угол)

Таким образом, по окончанию работы алгоритма имеем заполнение квадрата, представленное на рис. 5:

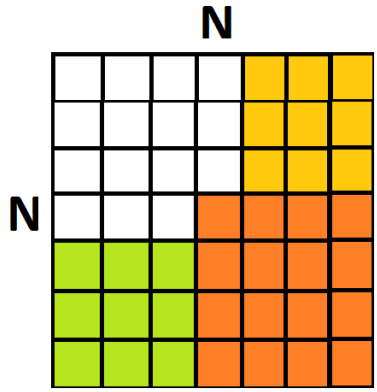


Рисунок 5 - Применение первой оптимизации

Вторая оптимизация:

В процессе работы функции, осуществляющей бэктрекинг, описанной ниже, не имеет смысла продолжать рассмотрение замощений, которые дают заведомо худшее решение. Поэтому заводится переменная minCount, равная минимальному из уже полученных числу разбиений квадрата, изначально равная N*N+1. Таким образом, количество квадратов текущего разбиения сравнивается с этой переменной, и в случае, если второе значение больше minCount, то процесс добавления элементов прекращается, происходит откат

назад и рассматривается следующий случай, иначе minCount присваивается количество квадратов текущего разбиения.

Описание функции backTracking:

Итеративная функция int backTracking(unInt** table, unInt** res, unInt N, stack <SQUARE>* Zcur, stack <SQUARE>* Zopt); предназначенная для обработки незаполненного после оптимизации участка стола, принимает на вход два двумерных динамических массива: table (типа unsigned int), предназначенный для хранения текущего замощения стола квадратами, и res - предназначенный для хранения текущего минимального замощения стола, к концу работу функции в нем хранится результат работы. Кроме того, на вход подается значение N - размер стола, необходимый для пробегания по матрице, и два стека, имеющих тип структуры из трех полей:

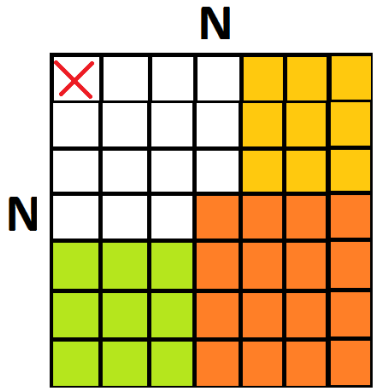
```
struct SQUARE {
	unInt x, y, size;	//координаты верхнего левого угла квадрата, длина его стороны
	};
```

В стек Zcur кладутся координаты (левого верхнего угла) и длины сторон добавляемых в матрицу элементов (квадратов), при удалении квадратов из текущего замощения, они также удаляются из стека. В стек Zopt копируются значения из стека Zcur, в случае нахождения нового минимального замощения. Использование двух массивов и двух стеков обоснованно тем, что во время перебора невозможно узнать, является ли новое минимальное замощение окончательным решением, таким образом, решения сохраняются.

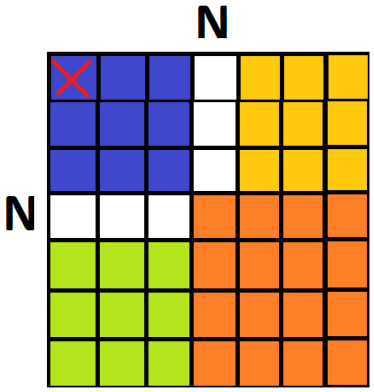
Алгоритм работы функции backTracking:

Для перебора всех возможных вариантов необходимо на каждой итерации выполнять последовательность действий:

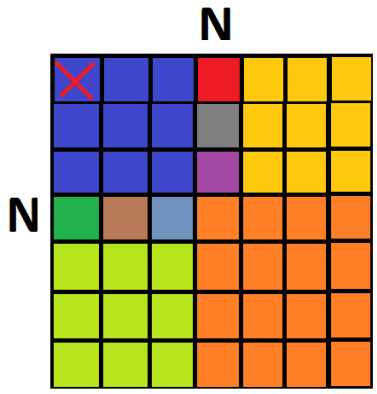
Шаг 0. Пробегаем по двумерному массиву (сверху вниз, слева направо) с помощью вложенных циклов в поиске еще не занятой позиции.



Шаг 1. Строим квадрат с максимально возможной стороной в этой позиции.



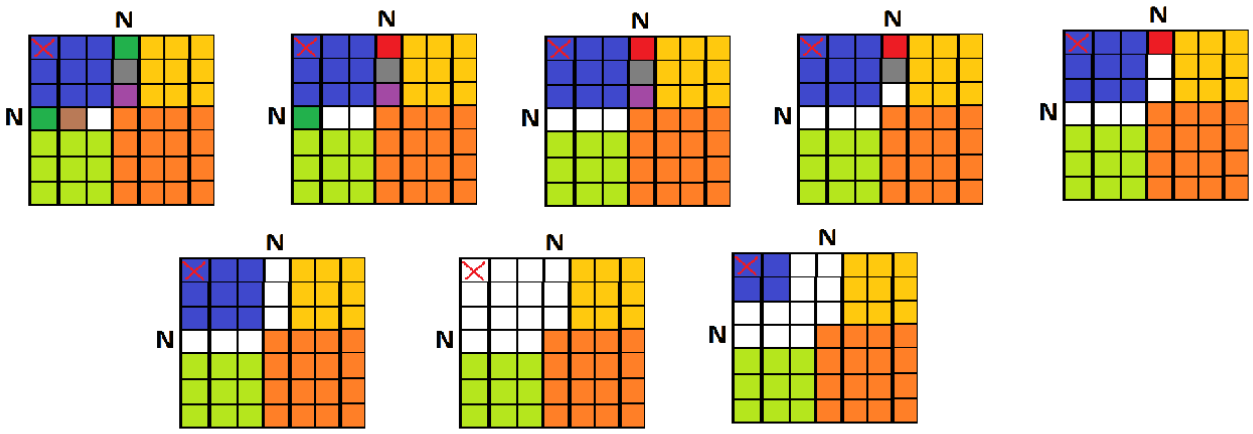
Шаг 2. Строим оставшиеся квадраты.



Шаг 3. Если полученное число квадратов текущего разбиения меньше предыдущего (при первой итерации всегда истинно), то сохраняем текущее разбиение в массив результата и соответствующие координаты и длины сторон в стек. Иначе сразу переходим к шагу 4.

Шаг 4. Производим откатывание назад: пробегая по массиву с конца (снизу вверх, справа налево) удаляем все элементы равных площадей. Дойдя до квадрата с большей стороной, например k, так же удаляем его и на той же

позиции строим квадрат со стороной k-1 (см. примечание).



Шаг 5. Возвращаемся к шагу 0, и продолжаем выполнение алгоритма, до тех пор пока длина стороны квадрата, построенного на самой первой позиции, не станет меньше одного.

Примечание. Смысл откатывания состоит в том, чтобы вернуться до момента, когда у нас был выбор (построить квадрат максимального размера или меньше) и проверить оставшиеся варианты. Таким образом, когда мы доходим до такого "особого" квадрата, появляется необходимость задавать его размер отдельно от остальных, позже построенных квадратов, так как иначе произойдет зацикливание алгоритма.

Описание дополнительных функций.

- Функция int dataCorrect(); ничего не принимает, необходима для проверки корректности введенных пользователем данных, так в случае введения строчных знаков, пользователь будет информирован об ошибке и невозможности продолжить выполнение программы до тех пор пока данные не будут введены корректно. Возвращает корректное значение, считанной переменной.
- Функция unInt** makeArray(unInt& N); принимает значение размера стола unInt& N. Необходима для создания двумерных

динамических массивов table и res, в которых хранятся текущее и минимальное разбиения соответственно. Функция ничего не возвращает.

- Функция void initArray(unInt& N, unInt** arr); принимает значение размера стола unInt& N и указатель на двумерный динамический массив unInt** arr. Нужна для инициализации полученного массива. Ничего не возвращает.
- Функция void printArray(unInt& N, unInt** arr); принимает значение размера стола unInt& N и указатель на двумерный динамический массив unInt** arr. Используется для печати двумерного массива. Функция ничего не возвращает.
- Функция void copeArray(unInt** arr1, unInt** arr2, unInt& N); принимает значение размера стола unInt& N и два указателя unInt** arr1, unInt** arr2 на двумерные динамические массивы. Используется для копирования содержимого из одной матрицы в другую. Функция ничего не возвращает.
- Функция void deleteArray(unInt& N, unInt** arr); принимает значение размера стола unInt& N и указатель на двумерный динамический массив unInt** arr. Используется для очистки памяти. Ничего не возвращает.
- Функция void printStack(stack <SQUARE>* a); принимает указатель на стек stack <SQUARE>* a, в котором хранится решение - координаты верхних левых углов квадратов, длины их сторон. Предназначена для печати содержимого стека. Ничего не возвращает.
- Функция void case1(unInt N, unInt** table, stack <SQUARE>* Zcur); принимает значение размера стола unInt N, указатель на двумерный динамический массив - стол, стек stack <SQUARE>* Zcur, в котором хранится частичное решение - координаты верхних левых

углов квадратов, длины их сторон. Предназначена для вывода решения первого частного случая. Ничего не возвращает.

- Функция void case2(unInt N, unInt** table, stack <SQUARE>* Zcur); принимает значение размера стола unInt N, указатель на двумерный динамический массив unInt** table - стол, стек stack <SQUARE>* Zcur, в котором хранится частичное решение - координаты верхних левых углов квадратов, длины их сторон. Предназначена для вывода решения второго частного случая. Ничего не возвращает.
- Функция void case3(unInt N, unInt** table, stack <SQUARE>* Zcur); принимает значение размера стола unInt N, указатель на двумерный динамический массив unInt** table - стол, стек stack <SQUARE>* Zcur, в котором хранится частичное решение - координаты верхних левых углов квадратов, длины их сторон. Предназначена для вывода решения третьего частного случая. Ничего не возвращает.
- Функция void training(unInt N, unInt** table, stack <SQUARE>* a); принимает значение размера стола unInt N, указатель на двумерный динамический массив unInt** table - стол, стек stack <SQUARE>* a, в которое записывается частичное решение после применения первой оптимизации - координаты верхних левых углов квадратов, длины их сторон. Нужна для применения первой оптимизации - рисования трех квадратов. Ничего не возвращает.
- Функция void paintSquare(unInt** table, unInt i, unInt j, unInt side, stack <SQUARE>* a); принимает указатель на матрицу unInt** table, координаты свободной позиции (unInt i, unInt j), размер стороны квадрата unInt side, который нужно построить, стек stack <SQUARE>* a, в котором хранятся текущие частичные решения - координаты верхних левых углов квадратов, длины их сторон. Используется для построения квадрата в нужной позиции с определенной стороной, а также для добавления координат и

стороны этого квадрата в стек частичных решений. Ничего не возвращает.

- Функция void deleteSquare(unInt i, unInt j, unInt side, unInt** table, stack <SQUARE>* a); принимает указатель на матрицу unInt** table, координаты позиции (unInt i, unInt j), начиная с которой нужно удалять квадрат, размер стороны квадрата unInt side, стек stack <SQUARE>* a, в котором хранятся текущие частичные решения - координаты верхних левых углов квадратов, длины их сторон.. Предназначена для удаления квадратов из матрицы и стека частичных решений. Ничего не возвращает.
- Функция unInt sizeSide(unInt** table, unInt i, unInt j, unInt side, unInt N); принимает указатель на матрицу unInt** table, координаты свободной позиции (unInt i, unInt j), размер стороны квадрата unInt side, значение размера стола unInt N. Предназначена для определения текущей максимально возможной стороны для квадрата, который будет построен в позиции с координатами (i; j) (построение начинается с левого верхнего угла). Ничего не возвращает.
- Булевая функция bool checkPos(unInt** table, unInt i, unInt j, unInt side, unInt N); принимает указатель на матрицу unInt** table, координаты, с которых начинается построение квадрата unInt i, unInt j, длину его стороны unInt side и значение размера стола unInt N. Вызывается функцией с одноименными параметрами unInt sizeSide(unInt** table, unInt i, unInt j, unInt side, unInt N); для проверки возможности построения квадрата в заданной позиции с выбранной стороной. Возвращает true, в случае, если квадрат возможно построить, и false в противном случае.

Также в программе используются несколько глобальных переменных

unInt g_covered; - площадь покрытия

unInt g_area; - площадь стола

unInt g_addColor; - хранит "цвет" добавленного квадрата (для цветного вывода в консоль)

Использование глобальных переменных не рекомендуется, в данном случае вынесены за пределы главной функции по причине частого использования в большинстве других функций.

Способы хранения частичных решений.

- Структура хранения координат верхнего левого угла квадрата и размера его стороны.

```
struct SQUARE {
    unInt x, y, size;
};
```

- Двумерные динамические массивы unInt** table и unInt** res типа unsigned int для хранения текущего и минимального замощений стола.
- Два стека stack <SQUARE>* curTilling и stack <SQUARE>* optTilling типа структуры, описанной выше, для хранения координат верхнего левого угла квадрата и размера его стороны каждого, включенного в замощение (curTilling - текущее, optTilling - минимальное) квадрата.

Оценка сложности алгоритма.

Сложность по памяти: можно оценить как O(N^2). Входные данные N - сторона квадрата, худшее разбиение N^2 - множество квадратов с единичными сторонами. Тогда на хранение двумерного массива и стека потребуется O($6N^2$), избавляясь от константы получаем O(N^2).

Сложность по времени является экспоненциальной. Внутри главного цикла, отвечающего за пробегание по двумерному массиву, сложность которого равна $O\left(\left(\frac{N}{2}\right)^2\right)$, отбросив константу получим $O(N^2)$, происходят последовательные вызовы функций: unInt sizeSide(unInt** table, unInt i, unInt j, unInt side, unInt N), отвечающей за определение размера стороны квадрата, который можно добавить в текущее замощение без выхода за границы стола -

сложность которой равна $O(N^3)$, так как внутри цикла происходит вызов функции bool checkPos(unInt** table, unInt i, unInt j, unInt side, unInt N) - сложность равна $O(N^2)$; void deleteSquare(unInt i, unInt j, unInt side, unInt** table, stack <SQUARE>* a), используемая для удаления квадратов из замощения (откатывание назад) имеет сложность $O(N^2)$; void paintSquare(unInt** table, unInt i, unInt j, unInt side, stack <SQUARE>* a), необходимая для добавления квадратов в текущее замощение, также имеет сложность $O(N^2)$. Таким образом, так как все эти функции вызываются последовательно, а не вложенно, то общая сложность их вызовов равна $O(N^3)$, а общая сложность по времени тогда будет равна $O(N^6)$.

Тестирование.

Тест № 1 (расширенный).

Введите размер столешницы (sizeTable).

7

Стол, который нужно заполнить.

0 0 0 0 0 0 0

0 0 0 0 0 0 0

0 0 0 0 0 0 0

0 0 0 0 0 0 0

0 0 0 0 0 0 0

0 0 0 0 0 0 0

0 0 0 0 0 0 0

ШАГ 1: применение оптимизации.

Оптимизация - рисование первых трёх квадратов.

Рисование большого квадрата:

Вызов функции рисования.

На данный момент покрыто 32.6531 %.

0 0 0 0 0 0 0

0 0 0 0 0 0 0

0 0 0 0 0 0 0

0 0 0 1 1 1 1

0 0 0 1 1 1 1

0 0 0 1 1 1 1

0 0 0 1 1 1 1

Рисование левого соседнего квадрата:

Вызов функции рисования.

На данный момент покрыто 51.0204 %.

0 0 0 0 0 0 0

0 0 0 0 0 0 0

0 0 0 0 0 0 0

0 0 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

Рисование правого соседнего квадрата:

Вызов функции рисования.

На данный момент покрыто 69.3878 %.

0 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

ШАГ 2: запуск backtracking.

Итерация - 1

Стартовая длина стороны для добавляемого квадрата 4.

Найдена свободная позиция с координатами x = 1 y = 1.

Добавление квадрата со стороной 3.

Вызов функции рисования.

На данный момент покрыто 87.7551 %.

Полученное замощение:

4 4 4 0 3 3 3

4 4 4 0 3 3 3

4 4 4 0 3 3 3

0 0 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

Найдена свободная позиция с координатами x = 1 y = 4.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 89.7959 %.

Полученное замощение:

4 4 4 5 3 3 3

4 4 4 0 3 3 3

4 4 4 0 3 3 3

0 0 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

Найдена свободная позиция с координатами x = 2 y = 4.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 91.8367 %.

Полученное замощение:

4 4 4 5 3 3 3

4 4 4 6 3 3 3

4 4 4 0 3 3 3

0 0 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

Найдена свободная позиция с координатами x = 3 y = 4.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 93.8776 %.

Полученное замощение:

4 4 4 5 3 3 3

4 4 4 6 3 3 3

4 4 4 7 3 3 3

0 0 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

Найдена свободная позиция с координатами x = 4 y = 1.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 95.9184 %.

Полученное замощение:

4 4 4 5 3 3 3

4 4 4 6 3 3 3

4 4 4 7 3 3 3

8 0 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

Найдена свободная позиция с координатами x = 4 y = 2.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 97.9592 %.

Полученное замощение:

4 4 4 5 3 3 3

4 4 4 6 3 3 3

4 4 4 7 3 3 3

8 9 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

Найдена свободная позиция с координатами x = 4 y = 3.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 100 %.

Полученное замощение:

4 4 4 5 3 3 3

4 4 4 6 3 3 3

4 4 4 7 3 3 3

8 9 10 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

Итерация - 2

Новое оптимальное замощение: 10.

Удаление квадрата с позиции: x = 4, y = 3.

Вызов функции удаления.

На данный момент покрыто 97.9592 %.

4 4 4 5 3 3 3

4 4 4 6 3 3 3

4 4 4 7 3 3 3

8 9 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

Удаление квадрата с позиции: x = 4, y = 2.

Вызов функции удаления.

На данный момент покрыто 95.9184 %.

4 4 4 5 3 3 3

4 4 4 6 3 3 3

4 4 4 7 3 3 3

8 0 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

Удаление квадрата с позиции: x = 4, y = 1.

Вызов функции удаления.

На данный момент покрыто 93.8776 %.

4 4 4 5 3 3 3

4 4 4 6 3 3 3

4 4 4 7 3 3 3

0 0 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

Удаление квадрата с позиции: x = 3, y = 4.

Вызов функции удаления.

На данный момент покрыто 91.8367 %.

4 4 4 5 3 3 3

4 4 4 6 3 3 3

4 4 4 0 3 3 3

0 0 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

Удаление квадрата с позиции: x = 2, y = 4.

Вызов функции удаления.

На данный момент покрыто 89.7959 %.

4 4 4 5 3 3 3

4 4 4 0 3 3 3

4 4 4 0 3 3 3

0 0 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

Удаление квадрата с позиции: x = 1, y = 4.

Вызов функции удаления.

На данный момент покрыто 87.7551 %.

4 4 4 0 3 3 3

4 4 4 0 3 3 3

4 4 4 0 3 3 3

0 0 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

Удаление квадрата с позиции x = : 1, y = 1.

Вызов функции удаления.

На данный момент покрыто 69.3878 %.

0 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Добавление квадрата на позицию: x = 1, y = 1 со стороной: 2

Вызов функции рисования.

На данный момент покрыто 77.551 %.

4 4 0 0 3 3 3

4 4 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Итерация - 3

Стартовая длина стороны для добавляемого квадрата 4.

Найдена свободная позиция с координатами x = 1 y = 3.

Добавление квадрата со стороной 2.

Вызов функции рисования.

На данный момент покрыто 85.7143 %.

Полученное замощение:

4 4 5 5 3 3 3

4 4 5 5 3 3 3

0 0 0 0 3 3 3

0 0 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

Найдена свободная позиция с координатами x = 3 y = 1.

Добавление квадрата со стороной 2.

Вызов функции рисования.

На данный момент покрыто 93.8776 %.

Полученное замощение:

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

Найдена свободная позиция с координатами x = 3 y = 3.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 95.9184 %.

Полученное замощение:

4 4 5 5 3 3 3

4 4 5 5 3 3 3

6 6 7 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

Найдена свободная позиция с координатами x = 3 y = 4.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 97.9592 %.

Полученное замощение:

4 4 5 5 3 3 3

4 4 5 5 3 3 3

6 6 7 8 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

Найдена свободная позиция с координатами x = 4 y = 3.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 100 %.

Полученное замощение:

4 4 5 5 3 3 3

4 4 5 5 3 3 3

6 6 7 8 3 3 3

6 6 9 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

Итерация - 4

Новое оптимальное замощение: 9.

Удаление квадрата с позиции: x = 4, y = 3.

Вызов функции удаления.

На данный момент покрыто 97.9592 %.

4 4 5 5 3 3 3

4 4 5 5 3 3 3

6 6 7 8 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

Удаление квадрата с позиции: x = 3, y = 4.

Вызов функции удаления.

На данный момент покрыто 95.9184 %.

4 4 5 5 3 3 3

4 4 5 5 3 3 3

6 6 7 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

Удаление квадрата с позиции: x = 3, y = 3.

Вызов функции удаления.

На данный момент покрыто 93.8776 %.

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

Удаление квадрата с позиции x = : 3, y = 1.

Вызов функции удаления.

На данный момент покрыто 85.7143 %.

4 4 5 5 3 3 3

4 4 5 5 3 3 3

0 0 0 0 3 3 3

0 0 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

Добавление квадрата на позицию: x = 3, y = 1 со стороной: 1

Вызов функции рисования.

На данный момент покрыто 87.7551 %.

4 4 5 5 3 3 3

4 4 5 5 3 3 3

6 0 0 0 3 3 3

0 0 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

Итерация - 5

Стартовая длина стороны для добавляемого квадрата 4.

Найдена свободная позиция с координатами x = 3 y = 2.

Добавление квадрата со стороной 2.

Вызов функции рисования.

На данный момент покрыто 95.9184 %.

Полученное замощение:

4 4 5 5 3 3 3

4 4 5 5 3 3 3

6 7 7 0 3 3 3

0 7 7 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

Найдена свободная позиция с координатами x = 3 y = 4.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 97.9592 %.

Полученное замощение:

4 4 5 5 3 3 3

4 4 5 5 3 3 3

6 7 7 8 3 3 3

0 7 7 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

Найдена свободная позиция с координатами x = 4 y = 1.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 100 %.

Полученное замощение:

4 4 5 5 3 3 3

4 4 5 5 3 3 3

6 7 7 8 3 3 3

9 7 7 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

Итерация - 6

Текущее замощение превышает минимальное - прекращаем обработку данного случая.

Удаление квадрата с позиции: x = 4, y = 1.

Вызов функции удаления.

На данный момент покрыто 97.9592 %.

4 4 5 5 3 3 3

4 4 5 5 3 3 3

6 7 7 8 3 3 3

0 7 7 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

Удаление квадрата с позиции: x = 3, y = 4.

Вызов функции удаления.

На данный момент покрыто 95.9184 %.

4 4 5 5 3 3 3

4 4 5 5 3 3 3

6 7 7 0 3 3 3

0 7 7 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

Удаление квадрата с позиции x = : 3, y = 2.

Вызов функции удаления.

На данный момент покрыто 87.7551 %.

4 4 5 5 3 3 3

4 4 5 5 3 3 3

6 0 0 0 3 3 3

0 0 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

Добавление квадрата на позицию: x = 3, y = 2 со стороной: 1

Вызов функции рисования.

На данный момент покрыто 89.7959 %.

4 4 5 5 3 3 3

4 4 5 5 3 3 3

6 7 0 0 3 3 3

0 0 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

Итерация - 7

Стартовая длина стороны для добавляемого квадрата 4.

Найдена свободная позиция с координатами x = 3 y = 3.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 91.8367 %.

Полученное замощение:

4 4 5 5 3 3 3

4 4 5 5 3 3 3

6 7 8 0 3 3 3

0 0 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

Найдена свободная позиция с координатами x = 3 y = 4.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 93.8776 %.

Полученное замощение:

4 4 5 5 3 3 3

4 4 5 5 3 3 3

6 7 8 9 3 3 3

0 0 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

Итерация - 8

Текущее замощение превышает минимальное - прекращаем обработку данного случая.

Удаление квадрата с позиции: x = 3, y = 4.

Вызов функции удаления.

На данный момент покрыто 91.8367 %.

4 4 5 5 3 3 3

4 4 5 5 3 3 3

6 7 8 0 3 3 3

0 0 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

Удаление квадрата с позиции: x = 3, y = 3.

Вызов функции удаления.

На данный момент покрыто 89.7959 %.

4 4 5 5 3 3 3

4 4 5 5 3 3 3

6 7 0 0 3 3 3

0 0 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

Удаление квадрата с позиции: x = 3, y = 2.

Вызов функции удаления.

На данный момент покрыто 87.7551 %.

4 4 5 5 3 3 3

4 4 5 5 3 3 3

6 0 0 0 3 3 3

0 0 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

Удаление квадрата с позиции: x = 3, y = 1.

Вызов функции удаления.

На данный момент покрыто 85.7143 %.

4 4 5 5 3 3 3

4 4 5 5 3 3 3

0 0 0 0 3 3 3

0 0 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

Удаление квадрата с позиции x = : 1, y = 3.

Вызов функции удаления.

На данный момент покрыто 77.551 %.

4 4 0 0 3 3 3

4 4 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Добавление квадрата на позицию: x = 1, y = 3 со стороной: 1

Вызов функции рисования.

На данный момент покрыто 79.5918 %.

4 4 5 0 3 3 3

4 4 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Итерация - 9

Стартовая длина стороны для добавляемого квадрата 4.

Найдена свободная позиция с координатами x = 1 y = 4.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 81.6327 %.

Полученное замощение:

4 4 5 6 3 3 3

4 4 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Найдена свободная позиция с координатами x = 2 y = 3.

Добавление квадрата со стороной 2.

Вызов функции рисования.

На данный момент покрыто 89.7959 %.

Полученное замощение:

4 4 5 6 3 3 3

4 4 7 7 3 3 3

0 0 7 7 3 3 3

0 0 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

Найдена свободная позиция с координатами x = 3 y = 1.

Добавление квадрата со стороной 2.

Вызов функции рисования.

На данный момент покрыто 97.9592 %.

Полученное замощение:

4 4 5 6 3 3 3

4 4 7 7 3 3 3

8 8 7 7 3 3 3

8 8 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

Найдена свободная позиция с координатами x = 4 y = 3.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 100 %.

Полученное замощение:

4 4 5 6 3 3 3

4 4 7 7 3 3 3

8 8 7 7 3 3 3

8 8 9 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

Итерация - 10

Текущее замощение превышает минимальное - прекращаем обработку данного случая.

Удаление квадрата с позиции: x = 4, y = 3.

Вызов функции удаления.

На данный момент покрыто 97.9592 %.

4 4 5 6 3 3 3

4 4 7 7 3 3 3

8 8 7 7 3 3 3

8 8 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

Удаление квадрата с позиции x = : 3, y = 1.

Вызов функции удаления.

На данный момент покрыто 89.7959 %.

4 4 5 6 3 3 3

4 4 7 7 3 3 3

0 0 7 7 3 3 3

0 0 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

Добавление квадрата на позицию: x = 3, y = 1 со стороной: 1

Вызов функции рисования.

На данный момент покрыто 91.8367 %.

4 4 5 6 3 3 3

4 4 7 7 3 3 3

8 0 7 7 3 3 3

0 0 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

Итерация - 11

Стартовая длина стороны для добавляемого квадрата 4.

Найдена свободная позиция с координатами x = 3 y = 2.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 93.8776 %.

Полученное замощение:

4 4 5 6 3 3 3

4 4 7 7 3 3 3

8 9 7 7 3 3 3

0 0 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

Итерация - 12

Текущее замощение превышает минимальное - прекращаем обработку данного случая.

Удаление квадрата с позиции: x = 3, y = 2.

Вызов функции удаления.

На данный момент покрыто 91.8367 %.

4 4 5 6 3 3 3

4 4 7 7 3 3 3

8 0 7 7 3 3 3

0 0 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

Удаление квадрата с позиции: x = 3, y = 1.

Вызов функции удаления.

На данный момент покрыто 89.7959 %.

4 4 5 6 3 3 3

4 4 7 7 3 3 3

0 0 7 7 3 3 3

0 0 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

Удаление квадрата с позиции x = : 2, y = 3.

Вызов функции удаления.

На данный момент покрыто 81.6327 %.

4 4 5 6 3 3 3

4 4 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Добавление квадрата на позицию: x = 2, y = 3 со стороной: 1

Вызов функции рисования.

На данный момент покрыто 83.6735 %.

4 4 5 6 3 3 3

4 4 7 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Итерация - 13

Стартовая длина стороны для добавляемого квадрата 4.

Найдена свободная позиция с координатами x = 2 y = 4.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 85.7143 %.

Полученное замощение:

4 4 5 6 3 3 3

4 4 7 8 3 3 3

0 0 0 0 3 3 3

0 0 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

Найдена свободная позиция с координатами x = 3 y = 1.

Добавление квадрата со стороной 2.

Вызов функции рисования.

На данный момент покрыто 93.8776 %.

Полученное замощение:

4 4 5 6 3 3 3

4 4 7 8 3 3 3

9 9 0 0 3 3 3

9 9 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

Итерация - 14

Текущее замощение превышает минимальное - прекращаем обработку данного случая.

Удаление квадрата с позиции x = : 3, y = 1.

Вызов функции удаления.

На данный момент покрыто 85.7143 %.

4 4 5 6 3 3 3

4 4 7 8 3 3 3

0 0 0 0 3 3 3

0 0 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

Добавление квадрата на позицию: x = 3, y = 1 со стороной: 1

Вызов функции рисования.

На данный момент покрыто 87.7551 %.

4 4 5 6 3 3 3

4 4 7 8 3 3 3

9 0 0 0 3 3 3

0 0 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

Итерация - 15

Стартовая длина стороны для добавляемого квадрата 4.

Уменьшение стартовой длины стороны квадрата = 3.

Итерация - 16

Текущее замощение превышает минимальное - прекращаем обработку данного случая.

Удаление квадрата с позиции: x = 3, y = 1.

Вызов функции удаления.

На данный момент покрыто 85.7143 %.

4 4 5 6 3 3 3

4 4 7 8 3 3 3

0 0 0 0 3 3 3

0 0 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

Удаление квадрата с позиции: x = 2, y = 4.

Вызов функции удаления.

На данный момент покрыто 83.6735 %.

4 4 5 6 3 3 3

4 4 7 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Удаление квадрата с позиции: x = 2, y = 3.

Вызов функции удаления.

На данный момент покрыто 81.6327 %.

4 4 5 6 3 3 3

4 4 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Удаление квадрата с позиции: x = 1, y = 4.

Вызов функции удаления.

На данный момент покрыто 79.5918 %.

4 4 5 0 3 3 3

4 4 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Удаление квадрата с позиции: x = 1, y = 3.

Вызов функции удаления.

На данный момент покрыто 77.551 %.

4 4 0 0 3 3 3

4 4 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Удаление квадрата с позиции x = : 1, y = 1.

Вызов функции удаления.

На данный момент покрыто 69.3878 %.

0 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Добавление квадрата на позицию: x = 1, y = 1 со стороной: 1

Вызов функции рисования.

На данный момент покрыто 71.4286 %.

4 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Итерация - 17

Стартовая длина стороны для добавляемого квадрата 4.

Найдена свободная позиция с координатами x = 1 y = 2.

Добавление квадрата со стороной 3.

Вызов функции рисования.

На данный момент покрыто 89.7959 %.

Полученное замощение:

4 5 5 5 3 3 3

0 5 5 5 3 3 3

0 5 5 5 3 3 3

0 0 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

Найдена свободная позиция с координатами x = 2 y = 1.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 91.8367 %.

Полученное замощение:

4 5 5 5 3 3 3

6 5 5 5 3 3 3

0 5 5 5 3 3 3

0 0 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

Найдена свободная позиция с координатами x = 3 y = 1.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 93.8776 %.

Полученное замощение:

4 5 5 5 3 3 3

6 5 5 5 3 3 3

7 5 5 5 3 3 3

0 0 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

Найдена свободная позиция с координатами x = 4 y = 1.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 95.9184 %.

Полученное замощение:

4 5 5 5 3 3 3

6 5 5 5 3 3 3

7 5 5 5 3 3 3

8 0 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

Найдена свободная позиция с координатами x = 4 y = 2.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 97.9592 %.

Полученное замощение:

4 5 5 5 3 3 3

6 5 5 5 3 3 3

7 5 5 5 3 3 3

8 9 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

Итерация - 18

Текущее замощение превышает минимальное - прекращаем обработку данного случая.

Удаление квадрата с позиции: x = 4, y = 2.

Вызов функции удаления.

На данный момент покрыто 95.9184 %.

4 5 5 5 3 3 3

6 5 5 5 3 3 3

7 5 5 5 3 3 3

8 0 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

Удаление квадрата с позиции: x = 4, y = 1.

Вызов функции удаления.

На данный момент покрыто 93.8776 %.

4 5 5 5 3 3 3

6 5 5 5 3 3 3

7 5 5 5 3 3 3

0 0 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

Удаление квадрата с позиции: x = 3, y = 1.

Вызов функции удаления.

На данный момент покрыто 91.8367 %.

4 5 5 5 3 3 3

6 5 5 5 3 3 3

0 5 5 5 3 3 3

0 0 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

Удаление квадрата с позиции: x = 2, y = 1.

Вызов функции удаления.

На данный момент покрыто 89.7959 %.

4 5 5 5 3 3 3

0 5 5 5 3 3 3

0 5 5 5 3 3 3

0 0 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

Удаление квадрата с позиции x = : 1, y = 2.

Вызов функции удаления.

На данный момент покрыто 71.4286 %.

4 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Добавление квадрата на позицию: x = 1, y = 2 со стороной: 2

Вызов функции рисования.

На данный момент покрыто 79.5918 %.

4 5 5 0 3 3 3

0 5 5 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Итерация - 19

Стартовая длина стороны для добавляемого квадрата 4.

Найдена свободная позиция с координатами x = 1 y = 4.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 81.6327 %.

Полученное замощение:

4 5 5 6 3 3 3

0 5 5 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Найдена свободная позиция с координатами x = 2 y = 1.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 83.6735 %.

Полученное замощение:

4 5 5 6 3 3 3

7 5 5 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Найдена свободная позиция с координатами x = 2 y = 4.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 85.7143 %.

Полученное замощение:

4 5 5 6 3 3 3

7 5 5 8 3 3 3

0 0 0 0 3 3 3

0 0 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

Найдена свободная позиция с координатами x = 3 y = 1.

Добавление квадрата со стороной 2.

Вызов функции рисования.

На данный момент покрыто 93.8776 %.

Полученное замощение:

4 5 5 6 3 3 3

7 5 5 8 3 3 3

9 9 0 0 3 3 3

9 9 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

Итерация - 20

Текущее замощение превышает минимальное - прекращаем обработку данного случая.

Удаление квадрата с позиции x = : 3, y = 1.

Вызов функции удаления.

На данный момент покрыто 85.7143 %.

4 5 5 6 3 3 3

7 5 5 8 3 3 3

0 0 0 0 3 3 3

0 0 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

Добавление квадрата на позицию: x = 3, y = 1 со стороной: 1

Вызов функции рисования.

На данный момент покрыто 87.7551 %.

4 5 5 6 3 3 3

7 5 5 8 3 3 3

9 0 0 0 3 3 3

0 0 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

Итерация - 21

Стартовая длина стороны для добавляемого квадрата 4.

Уменьшение стартовой длины стороны квадрата = 3.

Итерация - 22

Текущее замощение превышает минимальное - прекращаем обработку данного случая.

Удаление квадрата с позиции: x = 3, y = 1.

Вызов функции удаления.

На данный момент покрыто 85.7143 %.

4 5 5 6 3 3 3

7 5 5 8 3 3 3

0 0 0 0 3 3 3

0 0 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

Удаление квадрата с позиции: x = 2, y = 4.

Вызов функции удаления.

На данный момент покрыто 83.6735 %.

4 5 5 6 3 3 3

7 5 5 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Удаление квадрата с позиции: x = 2, y = 1.

Вызов функции удаления.

На данный момент покрыто 81.6327 %.

4 5 5 6 3 3 3

0 5 5 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Удаление квадрата с позиции: x = 1, y = 4.

Вызов функции удаления.

На данный момент покрыто 79.5918 %.

4 5 5 0 3 3 3

0 5 5 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Удаление квадрата с позиции x = : 1, y = 2.

Вызов функции удаления.

На данный момент покрыто 71.4286 %.

4 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Добавление квадрата на позицию: x = 1, y = 2 со стороной: 1

Вызов функции рисования.

На данный момент покрыто 73.4694 %.

4 5 0 0 3 3 3

0 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Итерация - 23

Стартовая длина стороны для добавляемого квадрата 4.

Найдена свободная позиция с координатами x = 1 y = 3.

Добавление квадрата со стороной 2.

Вызов функции рисования.

На данный момент покрыто 81.6327 %.

Полученное замощение:

4 5 6 6 3 3 3

0 0 6 6 3 3 3

0 0 0 0 3 3 3

0 0 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

Найдена свободная позиция с координатами x = 2 y = 1.

Добавление квадрата со стороной 2.

Вызов функции рисования.

На данный момент покрыто 89.7959 %.

Полученное замощение:

4 5 6 6 3 3 3

7 7 6 6 3 3 3

7 7 0 0 3 3 3

0 0 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

Найдена свободная позиция с координатами x = 3 y = 3.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 91.8367 %.

Полученное замощение:

4 5 6 6 3 3 3

7 7 6 6 3 3 3

7 7 8 0 3 3 3

0 0 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

Найдена свободная позиция с координатами x = 3 y = 4.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 93.8776 %.

Полученное замощение:

4 5 6 6 3 3 3

7 7 6 6 3 3 3

7 7 8 9 3 3 3

0 0 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

Итерация - 24

Текущее замощение превышает минимальное - прекращаем обработку данного случая.

Удаление квадрата с позиции: x = 3, y = 4.

Вызов функции удаления.

На данный момент покрыто 91.8367 %.

4 5 6 6 3 3 3

7 7 6 6 3 3 3

7 7 8 0 3 3 3

0 0 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

Удаление квадрата с позиции: x = 3, y = 3.

Вызов функции удаления.

На данный момент покрыто 89.7959 %.

4 5 6 6 3 3 3

7 7 6 6 3 3 3

7 7 0 0 3 3 3

0 0 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

Удаление квадрата с позиции x = : 2, y = 1.

Вызов функции удаления.

На данный момент покрыто 81.6327 %.

4 5 6 6 3 3 3

0 0 6 6 3 3 3

0 0 0 0 3 3 3

0 0 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

Добавление квадрата на позицию: x = 2, y = 1 со стороной: 1

Вызов функции рисования.

На данный момент покрыто 83.6735 %.

4 5 6 6 3 3 3

7 0 6 6 3 3 3

0 0 0 0 3 3 3

0 0 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

Итерация - 25

Стартовая длина стороны для добавляемого квадрата 4.

Найдена свободная позиция с координатами x = 2 y = 2.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 85.7143 %.

Полученное замощение:

4 5 6 6 3 3 3

7 8 6 6 3 3 3

0 0 0 0 3 3 3

0 0 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

Найдена свободная позиция с координатами x = 3 y = 1.

Добавление квадрата со стороной 2.

Вызов функции рисования.

На данный момент покрыто 93.8776 %.

Полученное замощение:

4 5 6 6 3 3 3

7 8 6 6 3 3 3

9 9 0 0 3 3 3

9 9 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

Итерация - 26

Текущее замощение превышает минимальное - прекращаем обработку данного случая.

Удаление квадрата с позиции x = : 3, y = 1.

Вызов функции удаления.

На данный момент покрыто 85.7143 %.

4 5 6 6 3 3 3

7 8 6 6 3 3 3

0 0 0 0 3 3 3

0 0 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

Добавление квадрата на позицию: x = 3, y = 1 со стороной: 1

Вызов функции рисования.

На данный момент покрыто 87.7551 %.

4 5 6 6 3 3 3

7 8 6 6 3 3 3

9 0 0 0 3 3 3

0 0 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

Итерация - 27

Стартовая длина стороны для добавляемого квадрата 4.

Уменьшение стартовой длины стороны квадрата = 3.

Итерация - 28

Текущее замощение превышает минимальное - прекращаем обработку данного случая.

Удаление квадрата с позиции: x = 3, y = 1.

Вызов функции удаления.

На данный момент покрыто 85.7143 %.

4 5 6 6 3 3 3

7 8 6 6 3 3 3

0 0 0 0 3 3 3

0 0 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

Удаление квадрата с позиции: x = 2, y = 2.

Вызов функции удаления.

На данный момент покрыто 83.6735 %.

4 5 6 6 3 3 3

7 0 6 6 3 3 3

0 0 0 0 3 3 3

0 0 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

Удаление квадрата с позиции: x = 2, y = 1.

Вызов функции удаления.

На данный момент покрыто 81.6327 %.

4 5 6 6 3 3 3

0 0 6 6 3 3 3

0 0 0 0 3 3 3

0 0 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

Удаление квадрата с позиции x = : 1, y = 3.

Вызов функции удаления.

На данный момент покрыто 73.4694 %.

4 5 0 0 3 3 3

0 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Добавление квадрата на позицию: x = 1, y = 3 со стороной: 1

Вызов функции рисования.

На данный момент покрыто 75.5102 %.

4 5 6 0 3 3 3

0 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Итерация - 29

Стартовая длина стороны для добавляемого квадрата 4.

Найдена свободная позиция с координатами x = 1 y = 4.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 77.551 %.

Полученное замощение:

4 5 6 7 3 3 3

0 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Найдена свободная позиция с координатами x = 2 y = 1.

Добавление квадрата со стороной 3.

Вызов функции рисования.

На данный момент покрыто 95.9184 %.

Полученное замощение:

4 5 6 7 3 3 3

8 8 8 0 3 3 3

8 8 8 0 3 3 3

8 8 8 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

Найдена свободная позиция с координатами x = 2 y = 4.

Добавление квадрата со стороной 1.

Вызов функции рисования.

На данный момент покрыто 97.9592 %.

Полученное замощение:

4 5 6 7 3 3 3

8 8 8 9 3 3 3

8 8 8 0 3 3 3

8 8 8 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

Итерация - 30

Текущее замощение превышает минимальное - прекращаем обработку данного случая.

Удаление квадрата с позиции: x = 2, y = 4.

Вызов функции удаления.

На данный момент покрыто 95.9184 %.

4 5 6 7 3 3 3

8 8 8 0 3 3 3

8 8 8 0 3 3 3

8 8 8 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

Удаление квадрата с позиции x = : 2, y = 1.

Вызов функции удаления.

На данный момент покрыто 77.551 %.

4 5 6 7 3 3 3

0 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Добавление квадрата на позицию: x = 2, y = 1 со стороной: 2

Вызов функции рисования.

На данный момент покрыто 85.7143 %.

4 5 6 7 3 3 3

8 8 0 0 3 3 3

8 8 0 0 3 3 3

0 0 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

Итерация - 31

Стартовая длина стороны для добавляемого квадрата 4.

Найдена свободная позиция с координатами x = 2 y = 3.

Добавление квадрата со стороной 2.

Вызов функции рисования.

На данный момент покрыто 93.8776 %.

Полученное замощение:

4 5 6 7 3 3 3

8 8 9 9 3 3 3

8 8 9 9 3 3 3

0 0 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

Итерация - 32

Текущее замощение превышает минимальное - прекращаем обработку данного случая.

Удаление квадрата с позиции x = : 2, y = 3.

Вызов функции удаления.

На данный момент покрыто 85.7143 %.

4 5 6 7 3 3 3

8 8 0 0 3 3 3

8 8 0 0 3 3 3

0 0 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

Добавление квадрата на позицию: x = 2, y = 3 со стороной: 1

Вызов функции рисования.

На данный момент покрыто 87.7551 %.

4 5 6 7 3 3 3

8 8 9 0 3 3 3

8 8 0 0 3 3 3

0 0 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

Итерация - 33

Стартовая длина стороны для добавляемого квадрата 4.

Уменьшение стартовой длины стороны квадрата = 3.

Итерация - 34

Текущее замощение превышает минимальное - прекращаем обработку данного случая.

Удаление квадрата с позиции: x = 2, y = 3.

Вызов функции удаления.

На данный момент покрыто 85.7143 %.

4 5 6 7 3 3 3

8 8 0 0 3 3 3

8 8 0 0 3 3 3

0 0 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

Удаление квадрата с позиции x = : 2, y = 1.

Вызов функции удаления.

На данный момент покрыто 77.551 %.

4 5 6 7 3 3 3

0 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Добавление квадрата на позицию: x = 2, y = 1 со стороной: 1

Вызов функции рисования.

На данный момент покрыто 79.5918 %.

4 5 6 7 3 3 3

8 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Итерация - 35

Стартовая длина стороны для добавляемого квадрата 4.

Найдена свободная позиция с координатами x = 2 y = 2.

Добавление квадрата со стороной 2.

Вызов функции рисования.

На данный момент покрыто 87.7551 %.

Полученное замощение:

4 5 6 7 3 3 3

8 9 9 0 3 3 3

0 9 9 0 3 3 3

0 0 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

Итерация - 36

Текущее замощение превышает минимальное - прекращаем обработку данного случая.

Удаление квадрата с позиции x = : 2, y = 2.

Вызов функции удаления.

На данный момент покрыто 79.5918 %.

4 5 6 7 3 3 3

8 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Добавление квадрата на позицию: x = 2, y = 2 со стороной: 1

Вызов функции рисования.

На данный момент покрыто 81.6327 %.

4 5 6 7 3 3 3

8 9 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Итерация - 37

Стартовая длина стороны для добавляемого квадрата 4.

Уменьшение стартовой длины стороны квадрата = 3.

Итерация - 38

Текущее замощение превышает минимальное - прекращаем обработку данного случая.

Удаление квадрата с позиции: x = 2, y = 2.

Вызов функции удаления.

На данный момент покрыто 79.5918 %.

4 5 6 7 3 3 3

8 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Удаление квадрата с позиции: x = 2, y = 1.

Вызов функции удаления.

На данный момент покрыто 77.551 %.

4 5 6 7 3 3 3

0 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Удаление квадрата с позиции: x = 1, y = 4.

Вызов функции удаления.

На данный момент покрыто 75.5102 %.

4 5 6 0 3 3 3

0 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Удаление квадрата с позиции: x = 1, y = 3.

Вызов функции удаления.

На данный момент покрыто 73.4694 %.

4 5 0 0 3 3 3

0 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Удаление квадрата с позиции: x = 1, y = 2.

Вызов функции удаления.

На данный момент покрыто 71.4286 %.

4 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

Вызов функции удаления.

На данный момент покрыто 69.3878 %.

Последний шаг: удаление квадрата с позиции: x = 1, y = 5.

0 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 0 0 3 3 3

0 0 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

РЕЗУЛЬТАТ:

4 4 5 5 3 3 3

4 4 5 5 3 3 3

6 6 7 8 3 3 3

6 6 9 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

Минимальное количество квадратов: 9

Координаты верхних углов квадратов и длины их сторон:

4 3 1

3 4 1

3 3 1

3 1 2

1 3 2

1 1 2

1 5 3

5 1 3

4 4 4

Время работы программы: 11 сек.

Если вы хотите продолжить работу с программой, введите любое положительное число, иначе - 0.

Таблица тестов

№	**Входная последовательность**	**Выходная последовательность**	**Время работы (секунд)**
1	4	Минимальное	1

		количество квадратов: 4 Координаты верхних углов квадратов и длины их сторон: 3 3 2 3 1 2 1 3 2 1 1 2	
2	8	Минимальное количество квадратов: 4 Координаты верхних углов квадратов и длины их сторон: 5 5 4 5 1 4 1 5 4 1 1 4	1
3	20	Минимальное количество квадратов: 4 Координаты верхних углов квадратов и длины их сторон: 11 11 10 11 1 10 1 11 10 1 1 10	3

4	3	Минимальное количество квадратов: 6 Координаты верхних углов квадратов и длины их сторон: 3 3 1 3 2 1 3 1 1 2 3 1 1 3 1 1 1 2	1
5	9	Минимальное количество квадратов: 6 Координаты верхних углов квадратов и длины их сторон: 7 7 3 7 4 3 7 1 3 4 7 3 1 7 3 1 1 6	1
6	27	Минимальное количество квадратов: 6 Координаты верхних углов квадратов и	2

		длины их сторон: 19 19 9 19 10 9 19 1 9 10 19 9 1 19 9 1 1 18	
7	5	Минимальное количество квадратов: 8 Координаты верхних углов квадратов и длины их сторон: 5 5 1 4 5 1 3 5 1 3 4 1 4 3 2 4 1 2 1 4 2 1 1 3	1
8	25	Минимальное количество квадратов: 8 Координаты верхних углов квадратов и длины их сторон: 21 21 5 16 21 5	4

		11 21 5 11 16 5 16 11 10 16 1 10 1 16 10 1 1 15	
9	6	Минимальное количество квадратов: 4 Координаты верхних углов квадратов и длины их сторон: 4 4 3 4 1 3 1 4 3 1 1 3	1
10	10	Минимальное количество квадратов: 4 Координаты верхних углов квадратов и длины их сторон: 6 6 5 6 1 5 1 6 5 1 1 5	1
11	15	Минимальное количество квадратов: 6	2

		Координаты верхних углов квадратов и длины их сторон: 11 11 5 11 6 5 11 1 5 6 11 5 1 11 5 1 1 1	
12	aaa	Ошибка. Данные некорректны! Ожидается положительное число.	---
13	-400	Ошибка. Данные некорректны! Ожидается положительное число.	---
14	13	Минимальное количество квадратов: 11 Координаты верхних углов квадратов и длины их сторон: 7 6 1 6 1 2 5 6 2 5 3 3	6

		4 3 1 4 1 2 1 4 4 1 1 3 1 8 6 8 1 6 7 7 7	
15	17	Минимальное количество квадратов: 12 Координаты верхних углов квадратов и длины их сторон: 9 8 1 7 8 2 7 5 3 6 5 1 6 1 4 5 8 2 5 6 2 1 6 4 1 1 5 1 10 8 10 1 8 9 9 9	7

Вывод.

В результате выполнения лабораторной работы была реализована программа с использованием итеративного бэктрекинга, вычисляющая

минимальное количество квадратов для разбиения квадрата размером N за разумное время.

ПРИЛОЖЕНИЕ А

```
#include <iostream> //доделать оформление+отчёт
#include <fstream>
#include <cmath>
#include <stack>
#include <ctime>
#define unInt unsigned int
using namespace std;

unInt g_covered;
//площадь покрытия
unInt g_area;
//площадь стола
unInt g_addColor;

struct SQUARE {
     unInt x, y, size;                          //координаты
верхнего левого угла квадрата, длина его стороны
};

unInt** makeArray(unInt& N);                    //функция      для
создания двумерного динамического массива?
void initArray(unInt& N, unInt** arr);      //инициализация
массива
void printArray(unInt& N, unInt** arr);     //печать массива
void copyArray(unInt** arr1, unInt** arr2, unInt& N);
void deleteArray(unInt& N, unInt** arr);    //очищение памяти
void printStack(stack <SQUARE>* a);           //для     печати
промежуточных результатов и ответа

void training(unInt N, unInt** table, stack <SQUARE>* a);
                                            //оптимизация
int backTracking(unInt** table, unInt** res, unInt N, stack
<SQUARE>* Zcur, stack <SQUARE>* Zopt); //бэктрекинг
```

```
    bool checkPos(unInt** table, unInt i, unInt j, unInt side,
unInt N);
    void paintSquare(unInt** table, unInt i, unInt j, unInt side,
stack <SQUARE>* a);          // для рисования квадратов
    unInt sizeSide(unInt** table, unInt i, unInt j, unInt side,
unInt N);
    void deleteSquare(unInt i, unInt j, unInt side, unInt**
table, stack <SQUARE>* a);             //удаление квадратов

    void case1(unInt N, unInt** table, stack <SQUARE>* Zcur);
        //кратно 2
    void case2(unInt N, unInt** table, stack <SQUARE>* Zcur);
        //кратно 3
    void case3(unInt N, unInt** table, stack <SQUARE>* Zcur);
        //кратно 5

    int dataCorrect();

    int main() {
        setlocale(LC_ALL, "rus");
        clock_t start, end;               //для подсчета времени
работы программы
        unInt exit = 1;
        stack <SQUARE>* curTilling = new stack <SQUARE>;
    //текущее замощение стола (координаты, длина сторон)
        stack <SQUARE>* optTilling = new stack <SQUARE>;
    //оптимальное замощение стола - оценка сверху

        cout << "\t\t\t\t\033[0;35mУСЛОВИЯ ЗАДАЧИ:\033[0m\n";
        //условия задачи
        cout << "    У Вовы много квадратных обрезков доски. Их
стороны изменяютнся от 1 до sizeTable-1, и у него есть \n
неограниченное число обрезков любого размера.";
```

```
        cout << " Но ему очень хочется получить большую столешницу \n - квадрат размера sizeTable. Он может получить ее, собрав из уже имеющихся обрезков(квадратов).\n";
        cout << " Внутри столешницы не должно быть пустот, обрезки не должны выходить за предеы столешницы и не должны \n перекрываться. Кроме того, Вова хочет использовать ";
        cout << "минимально возможное число обрезков.\n";
        cout << "\n   \033[0;35mВХОДНЫЕ ДАННЫЕ:\033[0m\n";
        cout << "     Размер столешницы - одно целое число sizeTable (2<=sizeTable<=30).\n";
        cout << "\n   \033[0;35mВЫХОДНЫЕ ДАННЫЕ:\033[0m\n";
        cout << "    Одно число minCount, задающее минимальное количество обрезков(квадратов), из которых можно \n построить столешницу(квадрат) заданного размера sizeTable.\n";
        cout << " Далее идет minCount строк, каждая из которых содержит три целых числа - координаты левого верхнего \n угла(1<=x, y<=sizeTable) и длину стороны соответствующего квадрата\n";

        while (exit) {      //для повторного использования
            start = clock();
            cout << "\n Введите размер столешницы (sizeTable).\n";
            unInt sizeTable = dataCorrect();
    //длина стороны квадрата

            unInt** table = makeArray(sizeTable);
        //двумерный массив - для хранения замощения стола
            initArray(sizeTable, table);
            unInt** res = makeArray(sizeTable);
        //двумерный массив - для хранения замощения стола
            initArray(sizeTable, res);

            g_covered = 0;                      //стол не покрыт
            g_area = pow(sizeTable, 2);
```

```
            cout << "Стол, который нужно заполнить.\n";
            printArray(sizeTable, table);
            cout << "\n";

            if (sizeTable % 2 == 0) {
                                        //кратно 2
                cout << "\033[0;33mЧастный случай: размер стола
кратен двум.\033[0m\n";
                cout << "Результат:\n";
                case1(sizeTable, table, curTilling);
            }
            else if (sizeTable % 3 == 0) {
                                        //кратно 3
                cout << "\033[0;33mЧастный случай: размер стола
кратен трём.\033[0m\n";
                cout << "Результат:\n";
                case2(sizeTable, table, curTilling);
            }
            else if (sizeTable % 5 == 0) {
                                        //кратно 5
                cout << "\033[0;33mЧастный случай: размер стола
кратен пяти.\033[0m\n";
                cout << "Результат:\n";
                case3(sizeTable, table, curTilling);
            }
            else {
                                                //N   -   простое
число
                cout    <<    "\033[0;33mШАГ    1:    применение
оптимизации.\033[0m\n";
                cout << "\n";
                training(sizeTable, table, curTilling);
```

```
                    cout     <<     "\033[0;33mШАГ     2:     запуск
backtracking.\033[0m\n";
                    cout << "\n";
                    backTracking(table, res, sizeTable, curTilling,
optTilling);
               }

               deleteArray(sizeTable, table);
               end = clock();
               cout << "\033[0;32mВремя работы программы: " << (end
- start) / CLK_TCK << " сек.\033[0m\n";
               cout << "\033[0;36mЕсли вы хотите продолжить работу
с программой, введите любое положительное число, иначе -
0.\033[0m\n";
               exit = dataCorrect();
          }
          system("pause");
          return 0;
     }

     int dataCorrect() { //функция, проверяющая корректность
входных данных
          int number = 0;

          while (!(cin >> number) || number < 0) {
               cout  <<  "\033[0;31mОшибка.  Данные  некорректны!
Ожидается положительное число.\033[0m" << endl;
               cin.clear(); //сброс коматозного состояние cin
               cin.ignore(1000, '\n');
               fflush(stdin); //очистка потока ввода
          }

          cin.get();
          return number;
     }
```

```
void case1(unInt N, unInt** table, stack <SQUARE>* Zcur) {
//если кратно 2
     unInt wSquare = N / 2;

     paintSquare(table, 0, 0, wSquare, Zcur);
//сохраняем в стек
     printArray(N, table);
//печатаем каждое добавление квадрата

     paintSquare(table, 0, wSquare, wSquare, Zcur);
     printArray(N, table);

     paintSquare(table, wSquare, 0, wSquare, Zcur);
     printArray(N, table);

     paintSquare(table, wSquare, wSquare, wSquare, Zcur);
     printArray(N, table);
     printStack(Zcur);
     //результат
}

void case2(unInt N, unInt** table, stack <SQUARE>* Zcur) {
     //кратно 3
     unInt wBigSq = (2 * N) / 3;
     unInt wSmallSq = N - wBigSq;

     paintSquare(table, 0, 0, wBigSq, Zcur);
     printArray(N, table);

     paintSquare(table, 0, wBigSq, wSmallSq, Zcur);
     printArray(N, table);

     paintSquare(table, wSmallSq, wBigSq, wSmallSq, Zcur);
     printArray(N, table);
```

```
        paintSquare(table, wBigSq, 0, wSmallSq, Zcur);
        printArray(N, table);

        paintSquare(table, wBigSq, wSmallSq, wSmallSq, Zcur);
        printArray(N, table);

        paintSquare(table, wBigSq, wBigSq, wSmallSq, Zcur);
        printArray(N, table);
        printStack(Zcur);
    }

    void case3(unInt N, unInt** table, stack <SQUARE>* Zcur) {
//кратно 5
        unInt wBigSq = (3 * N) / 5;
        unInt wMedSq = N - wBigSq;
        unInt wSmallSq = wMedSq / 2;

        paintSquare(table, 0, 0, wBigSq, Zcur);
        printArray(N, table);

        paintSquare(table, 0, wBigSq, wMedSq, Zcur);
        printArray(N, table);

        paintSquare(table, wBigSq, 0, wMedSq, Zcur);
        printArray(N, table);

        paintSquare(table, wBigSq, wMedSq, wMedSq, Zcur);
        printArray(N, table);

        paintSquare(table, wMedSq, wBigSq, wSmallSq, Zcur);
        printArray(N, table);

        paintSquare(table, wMedSq, N - wSmallSq, wSmallSq, Zcur);
        printArray(N, table);
```

```
        paintSquare(table, wBigSq, N - wSmallSq, wSmallSq, Zcur);
        printArray(N, table);

        paintSquare(table, N - wSmallSq, N - wSmallSq, wSmallSq,
Zcur);
        printArray(N, table);
        printStack(Zcur);
    }

    void training(unInt N, unInt** table, stack <SQUARE>* a) {
//для оптимально и текущего замощений
        cout << "\033[0;32mОптимизация - рисование первых трёх
квадратов.\033[0m\n";
        unInt i, j;
        unInt side1 = (N / 2) + 1;                  //сторона
большого квадрата
        unInt side2 = N - side1;                    //два
средний квадрата

        cout       <<       "\033[0;36mРисование       большого
квадрата:\033[0m\n";
        paintSquare(table, N / 2, N / 2, side1, a); //рисуем
большой квадрат
        printArray(N, table);
        cout << "\n";

        cout    <<    "\033[0;36mРисование    левого    соседнего
квадрата:\033[0m\n";
        paintSquare(table, side1, 0, side2, a);//рисуем  средний
квадрата
        printArray(N, table);
        cout << "\n";
```

```
        cout << "\033[0;36mРисование правого соседнего
квадрата:\033[0m\n";
        paintSquare(table, 0, side1, side2, a);
        printArray(N, table);
        cout << "\n";
    }

    int backTracking(unInt** table, unInt** res, unInt N, stack
<SQUARE>* Zcur, stack <SQUARE>* Zopt) {
        ofstream fout("Test.txt", ios_base::app);
        unInt sizeSq = N - 1;;
        unInt check;
        unInt i, j;
        unInt iter = 0; //***
        unInt size;
        unInt minCount = N * N + 1;
        unInt flag = 0;

        while (Zcur->size() != g_area) {
           //пока стол не разбит на одиночные квадраты
             iter++;
             cout << "Итерация - " << iter << "\n";
             fout << "Итерация - " << iter << "\n";

             if (g_area != g_covered && !flag) {
             //пока не заполнили
                  sizeSq = (N / 2) + 1;
             //sizeSq - начинаем поиск с максимальной стороны

                  cout << "\033[0;36mСтартовая длина стороны для
добавляемого квадрата " << sizeSq << ".\033[0m\n";
                  fout << "Стартовая длина стороны для
добавляемого квадрата " << sizeSq << ".\n";
```

```
                    check = Zcur->size();
    //для проверки: добавился ли квадрат?
                    for (i = 0; i < N / 2 + 1; i++) {
                         for (j = 0; j < N / 2 + 1; j++) {

                              if (Zcur->size() >= minCount) {
    //если текущее разбиение, даёт результат хуже
                                   flag = 1;
                                   break;
                              }

                              if (!table[i][j]) {
    //если есть свободные позиции
                                   cout    <<    "\033[0;32mНайдена
свободная позиция с координатами x = " << i + 1 << " y = " << j +
1 << ".\033[0m\n";  //печать промежуточных значений
                                   fout   <<   "Найдена   свободная
позиция с координатами x = " << i + 1 << " y = " << j + 1 <<
".\n";
                                   size  =  sizeSide(table,  i,  j,
sizeSq, N);
                                   cout   <<   "\033[0;32mДобавление
квадрата со стороной " << size << ".\033[0m\n";
                                   fout << "Добавление квадрата со
стороной " << size << ".\n";

                                   paintSquare(table,  i,  j,  size,
Zcur);         //рисуем и добавляем в стек
                                   //печать промежуточных значений
                                   cout   <<   "\033[0;36mПолученное
замощение:\033[0m\n";
                                   fout << "Полученное замощение:";
                                   printArray(N, table);
                                   cout << "\n";
                              }
```

```
                    }
                }
                if (check == Zcur->size()) {
                    sizeSq--;
                    cout << "\033[0;31mУменьшение стартовой
длины стороны квадрата = " << sizeSq << ".\033[0m\n";
                    fout << "Уменьшение стартовой длины
стороны квадрата = " << sizeSq << ".\n";
                }
            }

            else if (g_area == g_covered || flag) {
    //если нет свободных позиций
                if (flag) cout << "\033[0;31mТекущее замощение
превышает минимальное - прекращаем обработку данного
случая.\033[0m\n";
                flag = 0;

                if (Zcur->size() < minCount) {
//если нашли лучшее замощение
                    *Zopt = *Zcur;
                    minCount = Zcur->size();
                    copyArray(table, res, N);
                    cout << "\033[0;32mНовое оптимальное
замощение: " << minCount << ".\033[0m\n";
                    fout << "Новое оптимальное замощение: " <<
minCount << ".\n";
                }

                while (!Zcur->empty() && Zcur->top().size == 1)
{         //удаляем все единичные блоки
                    if (Zcur->top().x == 1 && Zcur->top().y ==
1) {      //крайний случай
                        deleteSquare(Zcur->top().x - 1, Zcur-
>top().y - 1, Zcur->top().size, table, Zcur);
```

```
                                //печать результата
                                cout << "\033[0;31mПоследний шаг:
удаление квадрата с позиции: x = " << Zcur->top().x << ", y = " <<
Zcur->top().y << ".\033[0m\n";
                                fout << "Последний шаг: удаление
квадрата с позиции: x = " << Zcur->top().x << ", y = " << Zcur-
>top().y << ".\n";
                                printArray(N, table);
                                cout << "\n";
                                cout                                <<
"\033[0;32mРЕЗУЛЬТАТ:\033[0m\n";
                                fout << "РЕЗУЛЬТАТ:\n";
                                printArray(N, res);
                                cout << "\n";
                                printStack(Zopt);

                                return 0;           //завершение
                           }

                           cout << "\033[0;31mУдаление квадрата с
позиции: x = " << Zcur->top().x << ", y = " << Zcur->top().y <<
".\033[0m\n";
                           fout << "Удаление квадрата с позиции: x =
" << Zcur->top().x << ", y = " << Zcur->top().y << ".\n";

                           deleteSquare(Zcur->top().x - 1, Zcur-
>top().y - 1, Zcur->top().size, table, Zcur);

                                printArray(N, table);
                                cout << "\n";
                      }
                      if (!Zcur->empty()) {
                      //если есть, что удалять
                           unInt a = Zcur->top().x - 1;
     //сохраняем значения
```

```
                        unInt b = Zcur->top().y - 1;
                        sizeSq = Zcur->top().size - 1;

                        cout << "\033[0;31mУдаление квадрата с позиции x = : " << Zcur->top().x << ", y = " << Zcur->top().y << ".\033[0m\n";
                        fout << "Удаление квадрата с позиции x = : " << Zcur->top().x << ", y = " << Zcur->top().y << ".\n";

                        deleteSquare(a, b, Zcur->top().size, table, Zcur); //удаляем и создаем там же с размером на -1

                        printArray(N, table);
    //печать промежуточных значений
                        cout << "\n";
                        fout << "\n";
                        cout << "\033[0;32mДобавление квадрата на позицию: x = " << a + 1 << ", y = " << b + 1 << " со стороной: " << sizeSq << "\033[0m\n";
                        fout << "Добавление квадрата на позицию: x = " << a + 1 << ", y = " << b + 1 << " со стороной: " << sizeSq << "\n";

                        paintSquare(table, a, b, sizeSq, Zcur);
    //рисуем и добавляем в стек

                        printArray(N, table);
                        cout << "\n";
                        fout << "\n";
                    }
                }
            }
        }
```

```
     void deleteSquare(unInt i, unInt j, unInt side, unInt**
table, stack <SQUARE>* a) {              //для удаления квадратов
          cout << "\033[0;33mВызов функции удаления.\033[0m\n";
          g_covered -= pow(side, 2);              //уменьшаем
площадь покрытия
          cout << "На данный момент покрыто " << (double)g_covered
/ ((double)g_area / 100) << " %.\n";

          for (unInt x = i; x < i + side; x++) {
               for (unInt y = j; y < j + side; y++) {
                    table[x][y] = 0;
               }
          }
          a->pop();           //удаляем из стека
     }

     void paintSquare(unInt** table, unInt i, unInt j, unInt side,
stack <SQUARE>* a) {
          ofstream fout("Test.txt", ios_base::app);
          fout << "Вызов функции рисования.\n";
          cout << "\033[0;33mВызов функции рисования.\033[0m\n";
          g_covered += pow(side, 2);
          fout << "На данный момент покрыто " << (double)g_covered
/ ((double)g_area / 100) << " %.\n";
          cout << "На данный момент покрыто " << (double)g_covered
/ ((double)g_area / 100) << " %.\n";
          unInt color = a->size() + 1;

          for (unInt x = i; x < i + side; x++) {
               for (unInt y = j; y < j + side; y++) {
                    table[x][y] = color;
                    g_addColor = color;
               }
          }
```

```
            a->push(SQUARE{ i + 1, j + 1, side });
            fout.close();
      }

      unInt sizeSide(unInt** table, unInt i, unInt j, unInt side,
unInt N) { //какой квадрат можно вписать
            int flag = 0;
            for (int k = side; k > 0; k--) {
                  if (checkPos(table, i, j, k, N)) {
                        return k;
                        break;
                  }
            }
      }

      bool checkPos(unInt** table, unInt i, unInt j, unInt side,
unInt N) {          //можно ли нарисовать квадрат из выбранной
координаты
            unInt flag = 0;

            if ((i + side) <= N && (j + side) <= N) {         //не
заходит за границы
                  for (unInt x = i; x < i + side; x++) {
                        for (unInt y = j; y < j + side; y++) {
                              if (table[x][y]) {
                                    flag++;
                              }
                        }
                  }
            }

            else {
                  return flag;
            }
```

```
        if (flag) return false;       //возвращаем результат
        else return true;
    }

    unInt** makeArray(unInt& N) { //N - длина стороны стола
        unInt** arr = new unInt * [N];

        for (unInt i = 0; i < N; i++) {
            arr[i] = new unInt[N];
        }
        return arr;
    }

    void initArray(unInt& N, unInt** arr) {
    //инициализация массива нулями
        for (unInt i = 0; i < N; i++) {
            for (unInt j = 0; j < N; j++) {
                arr[i][j] = 0;
            }
        }
    }

    void printArray(unInt& N, unInt** arr) {
        ofstream fout("Test.txt", ios_base::app);

        for (unInt i = 0; i < N; i++) {
            for (unInt j = 0; j < N; j++) {
                if (arr[i][j] == g_addColor) {
                    cout  <<  "\033[0;327m"  <<  arr[i][j]  <<
"\033[0m" << " ";
                    fout << arr[i][j] << " ";
                }
                else {
                    cout << arr[i][j] << " ";
                    fout << arr[i][j] << " ";
```

```
                    }
               }
               cout << endl;
               fout << endl;
          }

          fout.close();
     }

     void copyArray(unInt** arr1, unInt** arr2, unInt& N) {
          for (unInt i = 0; i < N; i++) {
               for (unInt j = 0; j < N; j++) {
                    arr2[i][j] = arr1[i][j];
               }
          }
     }

     void printStack(stack <SQUARE>* a) {
          unInt size = a->size();
          cout << "\033[0;32mМинимальное количество квадратов: \033[0m" << size << "\n";
          cout << "\033[0;32mКоординаты верхних углов квадратов и длины их сторон:" << "\033[0m\n";

          for (unInt i = 0; i < size; i++) {
               cout << a->top().x << " " << a->top().y << " " << a->top().size << endl;
               a->pop();
          }
     }

     void deleteArray(unInt& N, unInt** arr) {
          for (unInt i = 0; i < N; i++) {
               delete[] arr[i];
          }
```

```
    delete[] arr;
}
```