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

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Построение и анализ алгоритмов»

Тема: Поиск с возвратом

Студентка гр. 9382 _____ Сорочина М.В.

Преподаватель _____ Фирсов М.А.

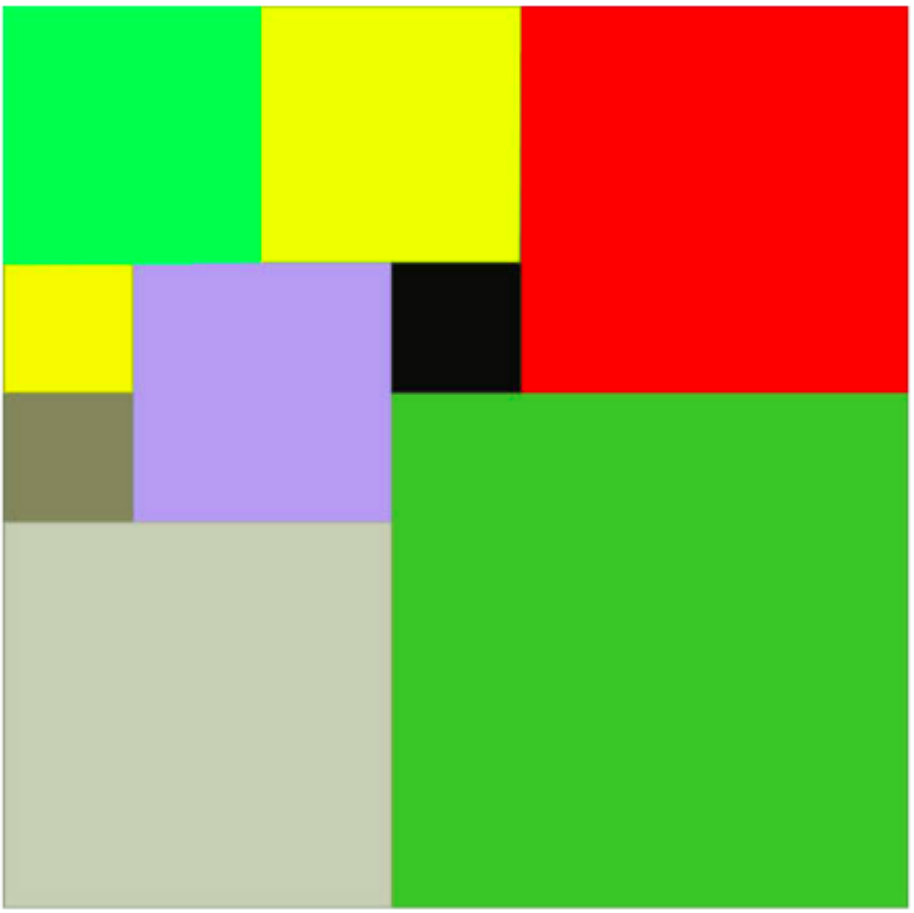
Санкт-Петербург

2021

Задание.

У Вовы много квадратных обрезков доски. Их стороны (размер) изменяются от 1 до $N-1$, и у него есть неограниченное число обрезков любого размера. Но ему очень хочется получить большую столешницу – квадрат размера N. Он может получить ее, собрав из уже имеющихся обрезков(квадратов).

Например, столешница размера 7×7 может быть построена из 9 обрезков.



Внутри столешницы не должно быть пустот, обрезки не должны выходить за пределы столешницы и не должны перекрываться. Кроме того, Вова хочет использовать минимально возможное число обрезков.

Входные данные

Размер столешницы – одно целое число N ($2 \le N \le 30$).

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

Вариант.

Вар. 3р. Рекурсивный бэктрекинг. Исследование кол-ва операций от размера квадрата.

Описание функций и структур данных.

1)
```
struct Square
{
   int x;
   int y;
   int w;
   Square(int a, int b, int c) : x(a), y(b), w(c) {}
};
```

Структура для хранения координат левого верхнего угла и длины стороны квадрата.

2) int divider(int n)

Функция поиска наименьшего делителя большего единицы. На вход принимает одно число - размер стороны квадрата. Возвращает делитель, если он имеется, и 1, если число простое.

3)void outputAns(int n, int m, std::vector <Square> sqrs)

Функция вывода ответа на экран. Принимает на вход два числа и вектор sqrs, в котором хранится список квадратов. n - размер квадрата, для которого производились подсчеты, m - второй множитель (для простого равен 1).

4)int isFull(std::vector<std::vector<int>> arr, int &y, int &x, int n, int &countOperations)

Функция проверки наличия свободных мест в квадрате, а также поиска нуля, если он имеется. Принимает на вход двумерный вектор arr, в котором хранятся актуальные данные о свободных и занятых местах в квадрате; ссылки на координаты x и y для записи координат нуля, при его наличии; n - размер квадрата; countOperations - переменная для хранения количества операций. Функция возвращает 1, если квадрат заполнен, и 0, если есть свободные места.

5)void printArr(std::vector<std::vector <int>> arr)

Функция вывода квадрата. Принимает на вход двумерный вектор arr, в котором хранится актуальное заполнение квадрата.

6)int maxWidth(int size, std::vector<std::vector<int>> arr, int i, int j, int &countOperations)

Функция поиска максимальной стороны квадрата для заданной точки. size - длина стороны квадрата, arr - двумерный массив, хранящий информацию о свободных и занятых местах квадрата, i и j - координаты, для которых нужно

найти максимальную длину, countOperations - переменная для хранения количества операций. Функция возвращает максимальную длину стороны квадрата.

7)void fill(std::vector<std::vector <int>> &arr, Square sqr)

Функция заполняющая квадрат значением его длины. arr - двумерный массив, хранящий информацию о свободных и занятых местах квадрата, sqr - экземпляр структуры square, в котором хранится информация о квадрате, который нужно добавить.

8)int squares(int &maxCount, std::vector<std::vector<int>> &arr, std::vector<Square> &sqrs, int n, int recLevel, int &countOperations)

Рекурсивная функция поиска минимального числа квадратов. maxCount - число, значение которого не меньше минимального количества квадратов (за вычетом трех); arr - двумерный массив, хранящий информацию о свободных и занятых местах квадрата; sqrs - экземпляр структуры Square, предназначенный для записи лучшего решения; n - размер квадрата; recLevel - уровень рекурсии; countOperations - переменная для хранения количества операций. Возвращает минимальное число квадратов.

Описание алгоритма.

Программа получает на вход одно число - длину стороны квадрата. Это число должно принадлежать промежутку от 2 до 40 включительно, иначе выводится сообщение о некорректности данных и программа завершается.

В случае, когда стороны квадрата составное число, находится его наименьший простой делитель и рассматривается квадрат со стороной, равной этому делителю. В противном случае, длина стороны не меняется.

Квадрат в памяти представлен в виде двумерного вектора arr. Для записи решения - вектор sqrs, size - размер квадрата после выявления делителя. В квадрат вставляются 3 квадрата: 1 со стороной size - size / 2 в точку (0, 0), 2

других, примыкающих к нему, со стороной size / 2. Данное разбиение занимает большую часть площади фигуры. Квадрат после вставки представлен на рис. 1.

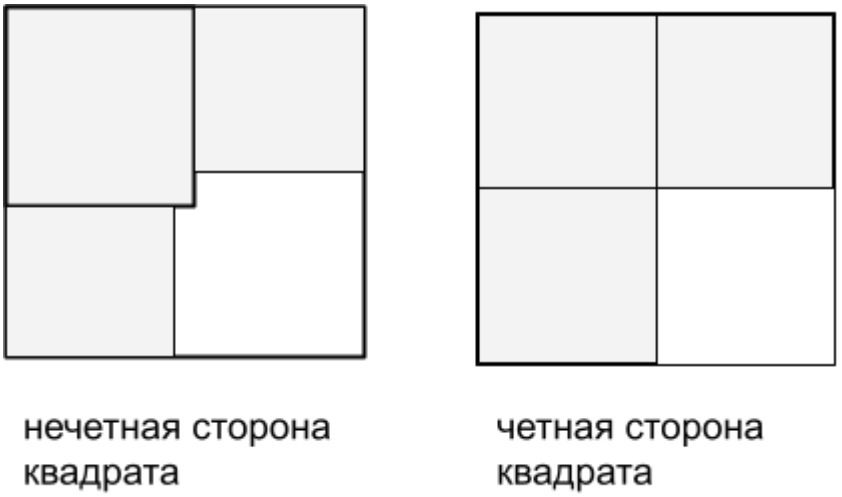


Рис. 1.

После этого запускается рекурсивная функция поиска лучшего решения. Первым делом проверяется сколько квадратов уже вставлено, чтобы отбросить разбиения, в которых много квадратов. После этого проверяется заполнен ли квадрат, если заполнен, то происходит возврат, перед которым в случае, если в этом разбиении меньше квадратов, чем в предыдущем лучшем, то переменная для хранения количества квадратов в лучшем разбиении меняется. Если в квадрате еще есть свободное место, то записываются координаты первого найденного нуля. Для этой точки ищется максимальная длина стороны квадрата, который туда можно вставить. Далее начиная с найденной длины и до 1 перебираются все варианты вставки.

В рекурсивной функции были созданы 2 вектора - std::vector<Square> current и std::vector<Square> best для записи частичных решений. current передается как аргумент в рекурсивную функцию, где в него записывается лучшее разбиение для тех данных, которые уже есть. Далее если полученное количество квадратов меньше ранее заданного minCount и больше 0, то в minCount записывается новое значение, в goodWidth записывается длина

стороны квадрата, который был вставлен, вектор best очищается и в него записываются значения из вектора current. Если же полученное число больше minCount или меньше единицы, то вектор current очищается. После этой проверки вставленный квадрат удаляется из arr.

После окончания цикла, вставляющего все возможные квадраты в данную точку, вектор sqrs, переданный в функцию, очищается и в него записываются значения из вектора best. Также в него добавляются данные ранее найденного квадрата, а именно: координаты точки, в которую на данном уровне рекурсии вставлялись квадраты, и лучшая длина стороны квадрата, которая была выявлена в ходе работы функции.

Использованные оптимизации:

1) Вставка в квадрат трех квадратов: 1 со стороной size - size / 2 в точку (0, 0), 2 других, примыкающих к нему, со стороной size / 2 (см. рис. 1).
2) Для составных чисел размер заменяется на наименьший простой делитель.
3) В случае, когда количество квадратов становится больше чем в зафиксированном ранее случае или в заданном изначально максимуме, разбиение дальше не рассматривается.

Оценка сложности.

По памяти: поскольку рассматривается квадрат со стороной n, то сложность O(n^2).

По времени: если убрать коэффициенты, которые ни на что не влияют, то получится, что мы рассматриваем квадрат со стороной n, где ищется пустая клетка (их n^2), затем вставляются квадраты со стороной от 1 до максимально возможной. Получается O(n^2*((n)!)^2).

Тестирование.

№ теста	Ввод	Вывод
1	2	4 2 2 1 1 1 1 2 1 1 1 2 1 Number of Operations is 9
2	3	6 3 3 1 2 3 1 3 2 1 1 1 2 3 1 1 1 3 1 Number of Operations is 34
3	4	4 3 3 2 1 1 2 3 1 2 1 3 2 Number of Operations is 9
4	5	8 5 5 1 4 5 1 3 5 1 3 4 1 4 3 2 1 1 3 4 1 2 1 4 2 Number of Operations is 231
5	7	9 6 6 2 4 6 2 7 5 1 4 5 1 7 4 1 5 4 2 1 1 4

		5 1 3 1 5 3 Number of Operations is 1181
6	9	6 7 7 3 4 7 3 7 4 3 1 1 6 7 1 3 1 7 3 Number of Operations is 34
7	11	11 9 9 3 6 9 3 11 8 1 10 8 1 6 8 1 6 7 1 10 6 2 7 6 3 1 1 6 7 1 5 1 7 5 Number of Operations is 28257
8	13	11 9 12 2 7 12 2 11 11 3 10 11 1 7 9 3 7 8 1 10 7 4 8 7 2 1 1 7 8 1 6 1 8 6 Number of Operations is 81903
9	15	6 11 11 5 6 11 5

		11 6 5 1 1 10 11 1 5 1 11 5 Number of Operations is 34
10	17	12 9 14 4 13 13 5 12 13 1 16 11 2 9 11 3 9 10 1 16 9 2 12 9 4 10 9 2 1 1 9 10 1 8 1 10 8 Number of Operations is 747764
11	19	13 13 17 3 10 17 3 16 16 4 15 16 1 14 16 1 10 13 4 10 12 1 10 11 1 14 10 6 11 10 3 1 1 10 11 1 9 1 11 9 Number of Operations is 2325219
12	23	13 19 19 5 19 17 2 12 17 7 21 16 3 20 16 1

		12 14 3 12 13 1 20 12 4 15 12 5 13 12 2 1 1 12 13 1 11 1 13 11 Number of Operations is 11682079
11	29	14 15 23 7 22 22 8 21 22 1 21 20 2 18 20 3 15 20 3 15 17 3 15 16 1 23 15 7 18 15 5 16 15 2 1 1 15 16 1 14 1 16 14 Number of Operations is 109724743

Исследование количества операций от размера квадрата.

В количестве операций учитывались те циклы и функции, которые больше всего вызываются во время работы программы. А именно: 1) количество попаданий в рекурсивную функцию, 2) количество проходов по циклам в функциях isFull, maxWidth, fill, printArr, а также количество проходов по циклу при удалении квадрата из массива. Эти операции оказывают наибольшее влияние на количество операции, остальные настолько малы, что ими можно пренебречь.

Зависимость количества операций от размера квадрата представлена на графике 1.

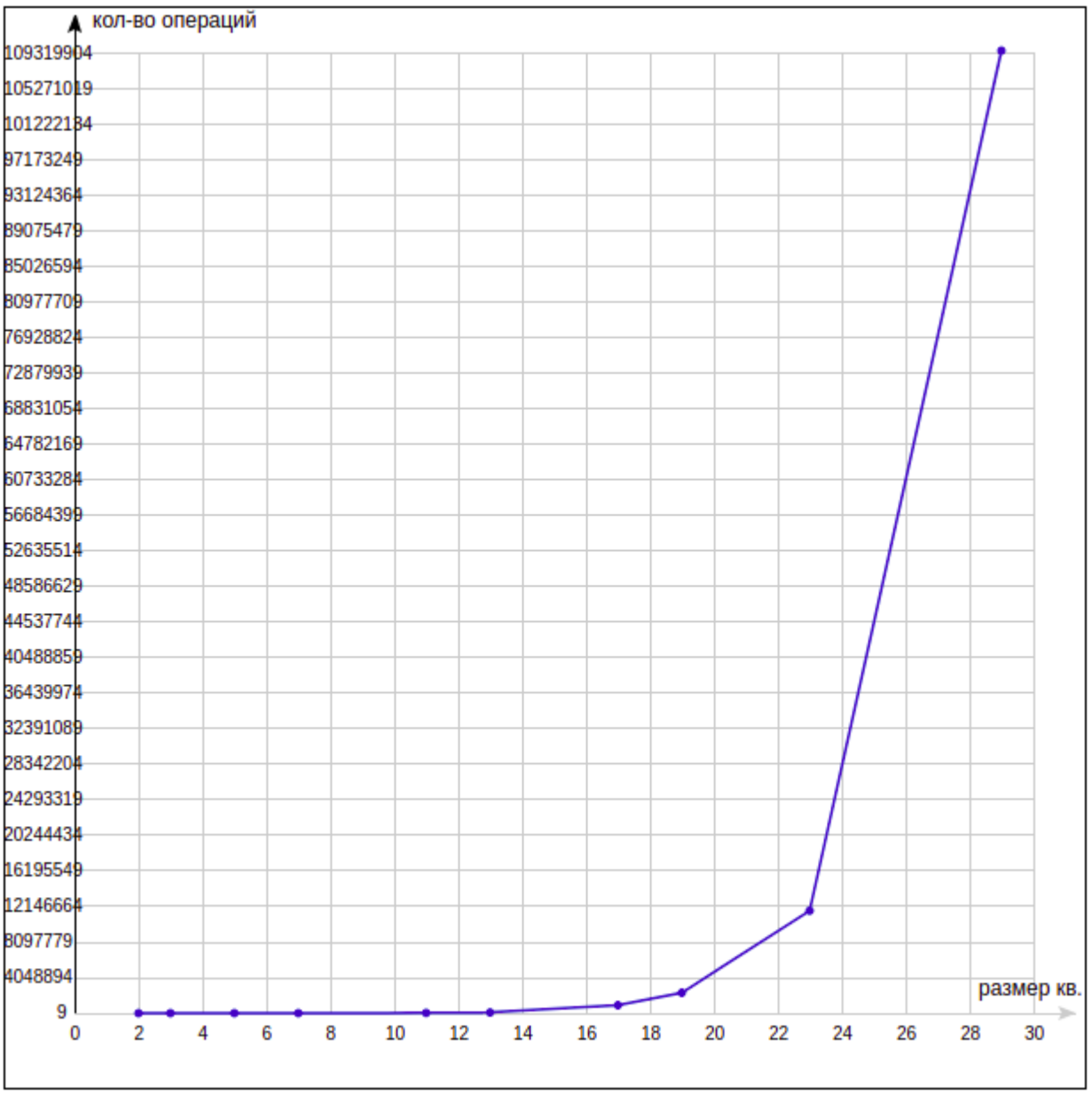


График 1.

Для всех составных чисел количество операций такое же, как для их наименьшего простого делителя, поэтому для удобства на графике представлены только простые числа.

Выводы.

В ходе выполнения работы была написана программа с рекурсивным бэктрекингом, а также было исследовано зависимость количества операций от размера квадрата.

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД ПРОГРАММЫ.

```
#include <iostream>
#include <vector>
#include <stack>
#include <cmath>

//#define NOTSTEPIK //выводит начальную фразу и кол-во операций
//#define INFO  //выводит сообщения, когда квадрат полон и рекурсия дошла до места, где нет смысла рассматривать дальше
//#define MOREINFO  //выводит сообщения, когда вставляется и удаляеься квадрат
//#define MOREINFOARR // "Расширение" для MOREINFO: выводит квадрат после изменений

struct Square
{       // структура для хранения координат и длины стороны
   int x; // x и y - координаты левой верхней точки
   int y;
   int w; // длина стороны
   Square(int a, int b, int c) : x(a), y(b), w(c) {}
};

//объявление функций
int divider(int);                                   //функция для поиска делителя
void outputAns(int, int, std::vector<Square>);              //функция вывода ответа
int isFull(std::vector<std::vector<int>>, int &, int &, int, int &); //функция проверки наличия свободных мест
void printArr(std::vector<std::vector<int>>);               //функция вывода квадрата
int maxWidth(int, std::vector<std::vector<int>>, int, int, int &);  //функция поиска максимального размера квадрата для заданных координат
void fill(std::vector<std::vector<int>> &, Square);                 //функция заполняющая квадрат значением его длины
int squares(int &, std::vector<std::vector<int>> &, std::vector<Square> &, int, int, int &);
//рекурсивная функция поиска решений

int main()
{
#ifdef NOTSTEPIK
   std::cout << "Please enter n\n";
#endif
   int n, countOperations = 0;
   std::cin >> n;
   if (n < 2 || n > 40)
   {
      std::cout << "incorreccountert input\n";
      return 0;
   }
   std::vector<Square> sqrs;
   int minDiv = divider(n), secondDiv = n, minimal;
   int size = (minDiv) == 1 ? (n) : (minDiv), numOfSqrs;
   std::vector<std::vector<int>> arr(size, std::vector<int>(size, 0));
   if (minDiv != 1)
```

```
    {
        minimal = 2 * size + 1;
        secondDiv /= minDiv;
    }
    else
    {
        minimal = n - n / 2 + 4;
    }
    fill(arr, {0, 0, size - size / 2});
    fill(arr, {size - size / 2, 0, size / 2});
    fill(arr, {0, size - size / 2, size / 2});
    int x;
    x = squares(minimal, arr, sqrs, size, 0, countOperations) + 3;
    sqrs.push_back({0, 0, size - size / 2});
    sqrs.push_back({size - size / 2, 0, size / 2});
    sqrs.push_back({0, size - size / 2, size / 2});
    outputAns(size, n / size, sqrs);
#ifdef NOTSTEPIK
    std::cout << "Number of Operations is " << countOperations << "\n";
#endif
    return 0;
}

int divider(int n)
{
    //функция для поиска делителя
    //возвращает наименьший простой делитель, иначе 1
    if (n % 2 == 0)
    {
        return 2;
    }
    if (n % 3 == 0)
    {
        return 3;
    }
    if (n % 5 == 0)
    {
        return 5;
    }
    return 1;
}

void outputAns(int n, int m, std::vector<Square> sqrs)
{
    //функция вывода ответа
    //n - размер, m - второй делитель (1, если простое)
    std::cout << sqrs.size() << '\n';
    for (auto i : sqrs)
    {
        std::cout << i.x * m + 1 << ' ' << i.y * m + 1 << ' ' << i.w * m << '\n';
    }
}
```

```
int isFull(std::vector<std::vector<int>> arr, int &y, int &x, int n, int &countOperations)
{
   //функция проверки наличия свободных мест
   //1 - нет мест, 0 - есть место
   //если есть ноль, то его координаты будут записаны в x и y
   for (x = n / 2; x < n; x++)
   {
      for (y = n / 2; y < n; y++)
      {
         countOperations++;
         if (arr[x][y] == 0)
         {
            return 0;
         }
      }
   }
   return 1;
}

void printArr(std::vector<std::vector<int>> arr)
{
   //функция вывода квадрата
   for (auto i : arr)
   {
      for (auto j : i)
      {
         std::cout << j << " ";
         if (j < 10)
         {
            std::cout << " ";
         }
      }
      std::cout << "\n";
   }
   std::cout << "\n";
}

int maxWidth(int size, std::vector<std::vector<int>> arr, int i, int j, int &countOperations)
{
   //функция поиска максимального размера квадрата для заданных координат i и j (y и x)
   int count = 1, xcount = 1, ycount = 1;
   while (j + xcount < size && arr[i][j + xcount] == 0)
   {
      xcount++;
   }
   while (i + ycount < size && arr[i + ycount][j] == 0)
   {
      ycount++;
   }
   countOperations += (xcount + ycount);
   count = (xcount) < (ycount) ? (xcount) : (ycount);
   return count;
}
```

```
void fill(std::vector<std::vector<int>> &arr, Square sqr)
{
   //функция заполняющая квадрат значением его длины
   for (int i = sqr.y; i < sqr.y + sqr.w; i++)
   {
      for (int j = sqr.x; j < sqr.x + sqr.w; j++)
      {
         arr[i][j] = sqr.w;
      }
   }
}

int squares(int &maxCount, std::vector<std::vector<int>> &arr, std::vector<Square> &sqrs, int n, int recLevel, int &countOperations)
{
   //рекурсивная функция поиска лучшего решения
   //возвращает кол-во квадратов
   countOperations++;
    if (maxCount < recLevel) //сравнение уровня рекурсии с макс. кол-вом квадратов, чтобы отбросить заведомо ненужные варианты
   {
#ifdef INFO
       std::cout << "Recursion level: " << recLevel << " maxCount = " << maxCount << ". Нет смысла дальше рассматривать.\n";
#endif
      return -1;
   }
   int x, y;
   if (isFull(arr, x, y, n, countOperations))
   {
      if (recLevel < maxCount)
      {
         maxCount = recLevel;
      }
#ifdef INFO
      std::cout << "Recursion level: " << recLevel << " maxCount = " << maxCount << ". Arr full.\n";
#endif
      return 0;
   }
   std::vector<Square> current, best;
   Square max(x, y, maxWidth(n, arr, y, x, countOperations));
   Square curr = max;
   int count, minCount = maxCount - recLevel, goodWidth = max.w;
    for (int width = max.w; width >= 1; width--) //цикл перебора размеров квадрата для заданных координат пустой точки
   {
      countOperations++;
      fill(arr, {max.x, max.y, width});
      countOperations += (width * width); // количество проходов по циклу в fill
#ifdef MOREINFO
      std::cout << "Recursion level: " << recLevel << " Сторона вставленного квадрата: " << width << ". Координаты: x = " << x << " y = " << y << ".\n";
```

```
#ifdef MOREINFOARR
      printArr(arr);
      countOperations += (n * n);
#endif
#endif
       count = squares(maxCount, arr, current, n, recLevel + 1, countOperations) + 1; //рекурсивный
вызов
      if (minCount > count && count > 0)
      {
          //если насчитали меньше чем было, то перезаписываем сторону квадрата и вектор,
хранящий разбиение
         minCount = count;
         goodWidth = width;
         best.clear();
         for (auto i : current)
         {
            best.push_back(i);
         }
      }
      else
      {
         //иначе очищаем текущий вектор
         current.clear();
      }
      for (int i = 0; i < width; i++) //удаление вставленного квадрата
      {
         for (int j = 0; j < width; j++)
         {
            arr[max.y + i][max.x + j] = 0;
         }
      }
      countOperations += (width * width); // количество проходов по циклу удаления
#ifdef MOREINFO
       std::cout << "Recursion level: " << recLevel << " Сторона удалённого квадрата: " << width << ".
Координаты: x = " << x << " y = " << y << ".\n";
#ifdef MOREINFOARR
      printArr(arr);
      countOperations += (n * n);
#endif
#endif
   }
   sqrs.clear();
   for (auto i : best)
   {
      sqrs.push_back(i);
   }
       sqrs.push_back({max.x, max.y, goodWidth}); //добавляем в точку с координатами x, y
квадрат длины goodWidth
   return minCount;
}
```