

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ**  
**по практической работе №1**  
**по дисциплине «Построение и анализ алгоритмов»**  
**Тема: Поиск с возвратом**

Студент гр. 9382

\_\_\_\_\_

Сорокумов С. В.

Преподаватель

\_\_\_\_\_

Фирсов М. А.

Санкт-Петербург

2021

### **Цель работы.**

Ознакомиться с алгоритмом перебора с возвратом и научиться применять его на практике. Написать программу, реализовывающую поиск с возвратом.

### **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  $7 \times 7$  может быть построена из 9 обрезков.



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

### **Входные данные**

Размер столешницы - одно целое число  $N$  ( $2 \leq N \leq 20$ ).

### **Выходные данные**

Одно число  $K$ , задающее минимальное количество обрезков(квадратов), из которых можно построить столешницу (квадрат) заданного размера  $N$ . Далее должны идти  $K$  строк, каждая из которых должна содержать три целых числа  $x$ ,  $y$  и  $w$ , задающие координаты левого верхнего угла ( $1 \leq x, y \leq N$ ) и длину стороны соответствующего обрезка (квадрата).

### **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

### **Вариант 1р.**

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

## Описание алгоритма.

На вход программе подается число от 2 до 40. Если число входит в заданный промежуток, то программа начинает работу, иначе программа сообщит об ошибке.

Если рассмотреть разбиение всех квадратов длины от 2 до 40, то можно вывести следующее утверждение: минимальному разбиению (разбиению с наименьшим количеством квадратов) для непростых чисел  $n$  (Напр.: 6, 21, 33) будет соответствовать разбиение квадрата с длиной стороны равной наименьшему целочисленному делителю числа  $n$  не равному единице. Так, с помощью функции `multiplicity` находится этот самый делитель `mul`, если он есть и далее идет работа уже либо с квадратом длины `mul`, либо с прежним квадратом, если наименьший делитель  $n$  равен единице.

Сам квадрат в памяти программы представлен в виде двумерного массива `squareArray`. Кроме того, существует стандартный стек `stackSquares` с типом элементов `Square`, где `Square` это структура, которая будет содержать информацию о квадрате, местоположение его левого верхнего угла относительно начала массива `squareArray` и длину самого квадрата.

Далее элементы `limit`, `& stackSquares`, `squareArray`,  $n$  (`mul`), 0 из `main` подаются в рекурсивную функцию поиска минимального количества квадратов, где 0 из этого списка означает 0-й уровень рекурсии.

В самой рекурсии происходит следующее:

Проверка на достижение предельной глубины рекурсии. Если это предел, то возвращается -1. Осуществляется поиск пустой клетки в массиве `squareArray`. Если таковой не оказалось, то возвращается 0. Создаются дополнительные стеки `stackSquaresTemp` и `stackSquaresMin`. В первый будут записываться текущие квадраты, а во второй их минимальная последовательность. Далее происходит нахождение длины до границы. Пробуются разные длины квадрата пока меньше чем расстояние до границы.

Для этого происходит заполнение квадрата функцией *fillSquare*. Рекурсивное обращение из функции в эту же функцию, только с счетчиком рекурсии +1.

Возвращенное значение количества квадратов записывается в переменную *k*.

Происходит проверка значения *k* на минимальное и, если это так, запоминаем длину текущего квадрата, *minCount* и переписываем *stackSquaresMin* значениями из *stackSquaresTemp*. Если же нет, то опустошаем *stackSquaresTemp*. Далее очищаем квадрат. И переходим к следующей длине квадрата. Когда вписать квадраты больше не получается, копируем в *stackSquares* значения из *stackSquaresMin*, записываем в *stackSquares* текущий квадрат и возвращаем *minCount*.

После выхода из рекурсии выводится *minCount*, значения квадратов в нужном масштабе и освобождается память.

Сложность алгоритма  $O(2^N)$ . Объяснить это можно тем, что для каждого маленького квадрата существует 2 варианта размещения: либо он ставится в большой квадрат, либо нет.

Сложность по памяти  $O(N^2)$ . Объясняется это тем, что в программе выделяется только память под массив размером  $N \times N$ .

### Описание структур.

Таблица 1 – Описание структур данных

Название структуры	Объект	Описание
struct Square { }	int x;	Координата квадрата x относительно начала массива.
	int y;	Координата квадрата y относительно начала массива.
	int length;	Длина квадрата.

## Описание функций.

Таблица 2 – Описание функций

Сигнатура	Параметры	Описание
<i>int multiplicity(int n)</i>	n – длина квадрата	Возвращает наименьший делитель не равный нулю, если это непростое число.  В случае с простым числом возвращает 1.
<i>void demonstration(int** squareArray, int n){</i>	<i>squareArray</i> – массив клеток квадрата  n – длина массива	Демонстрация массива.
<i>void fillSquare(int**squareArray, int x , int y, int lengthSquare)</i>	<i>squareArray</i> – массив клеток квадрата  x – координата x квадрата  y – координата y квадрата  <i>lengthSquare</i> – длина квадрата	Инициализирует квадрат в массиве в соответствии с параметрами x, y и <i>lengthSquare</i>
<i>int findEmptyCell(int**squareArray, int &amp;x, int &amp;y, int n)</i>	<i>squareArray</i> – массив клеток квадрата  x – координата x	Находит пустую клетку в массиве <i>squareArray</i> и переводит значения x и y в соответствии с ее

	квадрата у – координата у квадрата n – размер массива	координатами. Возвращает 1 в случае, когда нет пустых клеток, и 0 в обратном.
<i>oid stackCopy(std::stack          &lt;Square&gt; * stackSquares,          std::stack &lt;Square&gt; *          stackSquaresCopy)</i>	<i>stackSquares</i> – стек для копирования <i>stackSquaresCopy</i> – копируемый стек	Копирует стек.
<i>void          emptyStack(std::stack          &lt;Square&gt; *stackSquares)</i>	<i>stackSquares</i> – опустошаемый стек	Опустошает стек.
<i>int findMinSqrS(int limit,          std::stack &lt;Square&gt;          *stackSquares, int          **squareArray, int n, int          countRec, int scale)</i>	limit – предельное значение счетчика рекурсии <i>stackSquares</i> – стек для квадратов <i>squareArray</i> – массив клеток квадрата n – размер массива <i>countRec</i> – счетчик	Рекурсивная функция для перебора возможных значений расстановки квадратов.

### Тестирование.

Таблица 3 – Тестирование

№ теста	Тест	Результат
1	2	4

		1 1 1 1 2 1 2 1 1 2 2 1
2	6	4 1 1 3 1 4 3 4 1 3 4 4 3
3	13	11 1 1 7 1 8 6 8 1 6 7 8 1 7 9 3 7 12 2 8 7 2 9 12 2 10 7 4 10 11 1 11 11 3
4	51	Error
5	37	37 15 1 1 19 1 20 18 20 1 18 19 20 1 19 21 3 19 24 7



		19 31 7 20 19 2 22 19 5 26 24 2 26 26 12 27 19 4 27 23 1 28 23 3 31 19 7
6	25	8 1 1 5 1 6 5 1 11 5 1 16 10 6 1 10 6 11 5 11 11 15 16 1 10
7	33	6 1 1 11 1 12 11 1 23 11 12 1 11 12 12 22 23 1 11

### **Вывод.**

В ходе выполнения лабораторной работы был изучен и применен на практике алгоритм перебора с возвратом.

## ПРИЛОЖЕНИЕ А

### Исходный код программы

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

//Структура квадрата
struct Square{
    int x;
    int y;
    int length;
};

struct Point{
    int x;
    int y;
    int res;
};

//Проверка кратности
int multiplicity(int n){
    if(n % 2 == 0 && n != 2)
        return 2;
    if(n % 3 == 0 && n != 3)
        return 3;
    if(n % 5 == 0 && n != 5)
        return 5;
    return 1;
}

//Распечатка квадрата
void demonstration(int** squareArray, int n){
    std::cout << "Текущее состояние:" <<std::endl;
    for(int m = 0; m < n; m++){
        for(int l = 0; l < n; l++)
            std::cout << squareArray[m][l]<< ' ';
        std::cout << std::endl;
    }
    std::cout << std::endl;
}

//Проверяет возможность вместить квадрат определенной длины
int moreSquare(int **squareArray, int x, int y, int lengthSquare){
```

```

        for(int i = 0; i < lengthSquare; i++)
            if(squareArray[x + lengthSquare - 1][y + i] ||
squareArray[x + i][y + lengthSquare - 1])
                return 1;
        return 0;
    }

    //Раскрашивает квадрат
    void fillSquare(int **squareArray, int x , int y, int
lengthSquare){
        for(int i = 0; i < lengthSquare; i++)
            for(int j = 0; j < lengthSquare; j++)
                squareArray[x + i][y + j] = lengthSquare;
    }

    //Копирование стека
    void stackCopy(std::stack <Square> * stackSquares, std::stack
<Square> * stackSquaresCopy){
        while(!stackSquares->empty()){
            stackSquaresCopy->push(stackSquares->top());
            stackSquares->pop();
        }
    }

    // Нахождение пустой клетки
    Point findEmptyCell(int **squareArray, Point p, int n){
        for(p.x = 0; p.x < n; p.x++){
            for(p.y = 0; p.y < n - 1; p.y++){
                if(squareArray[p.x][p.y] == 0)
                    break;
                if(squareArray[p.x][p.y] == 0)
                    break;
            }
            if(p.x == n){
                p.x = n - 1;
            }
            if(squareArray[p.x][p.y] != 0) {
                p.res = 1;
                return p;
            }
            p.res=0;
            return p;
        }
    }

```

```

//Опустащает стек
void emptyStack(std::stack <Square> *stackSquares){
    while(!stackSquares->empty())
        stackSquares->pop();
}

//Очищает квадрат
void clearSquare(int **squareArray, int x, int y, int
lengthSquare){
    for(int i = 0; i < lengthSquare; i++)
        for(int j = 0; j < lengthSquare; j++)
            squareArray[x+i][y+j] = 0;
}

//Находит рекурсивно минимальное распделение квадрата
int findMinSqrs(int limit, std::stack <Square> *stackSquares, int
**squareArray, int n, int countRec, int scale){
    if(limit < countRec){ //Условие выхода за пределы
        std::cout << std::endl << std::endl << "Выход за пределы."
<<std:: endl;
        return -1;
    } else{
        std::cout << "Поставим следующий квадрат." << std::endl;
    }
    demonstration(squareArray, n); //Вывод заполненности квадрата
    int x, y;
    Point p;
    p = findEmptyCell(squareArray, p, n);
    x = p.x;
    y = p.y;
    if(p.res) // Нахождение пустой клетки
    {
        std::cout << "Пустых клеток не осталось." << std::endl;
        return 0;
    }else{
        std::cout << "Найдена пустая клетка: x = " << x << ", y =
"<< y << std::endl;
    }
    std::stack <Square> stackSquaresTemp; //Временный стек
квадратов
    std::stack <Square> stackSquaresMin; //Итоговый стек квадратов
    int distanceToTheBorder = (n - x > n - y) ? (n - y -
!squareArray[0][0]) : (n - x - !squareArray[0][0]); //Хранит расстояние
до границы

```

```

        std::cout << "Расстояние до границ = " << distanceToTheBorder
<< std::endl;
        int lengthSquare;
        int minCount = limit + 1;
        int k;
        int need_length = 1;
        std::cout << "Перебер длин квадрата:" << std::endl;
        for(lengthSquare = 1; lengthSquare <= distanceToTheBorder;
lengthSquare++){ //Пробегает по длине квадрата
            std::cout << "Текущая длина: " << lengthSquare <<
std::endl;

            std::cout << "Заполним квадрат. ";
            fillSquare(squareArray, x, y, lengthSquare);//Заполняет
квадрат

            k = findMinSqrS(limit, &stackSquaresTemp, squareArray, n,
countRec + 1, scale) + 1;
            if(k < minCount && k != 0){ //Если найдено распределение с
меньшим количеством квадратов сохраним стек квадратов
                std::cout << "Нашли распределение лучше, чем " << limit
+ 1 << ". Сохраним его." << std::endl;
                minCount = k; //Сохраниаем минимальное распределение
                need_length = lengthSquare;
                emptyStack(&stackSquaresMin);
                stackCopy(&stackSquaresTemp, &stackSquaresMin);
            }
            else {//Иначе очищаем временный стек
                std::cout << "Было найдено распределение по-лучше,
очистим стек квадратов." << std::endl;
                emptyStack(&stackSquaresTemp);
            }
            std::cout << "Сотрем квадрат длиной = " << lengthSquare
<< " в точке x = " << x << ", y = " << y << "." << std::endl;
            clearSquare(squareArray, x, y, lengthSquare);//Стираем
нарисованный квадрат
            demonstration(squareArray, n);

            if(distanceToTheBorder - lengthSquare - 1 < 0) break;
            if(moreSquare(squareArray, x, y, lengthSquare + 1)) // Если
нельзя вместить квадрат побольше, выходим
            {
                std::cout << "Квадрат побольше вместить не
получилось." << std::endl;
                break;
            }

```

```

    }
    emptyStack(stackSquares);
    stackCopy(&stackSquaresMin, stackSquares); //Копируем стек в
основной стек
    stackSquares->push({x * scale + 1, y * scale + 1, need_length
* scale});
    return minCount;
}

void deleteArray(int** array, int n){
    for(int i = 0; i < n; i++)
        delete [] array[i];
    delete array;
}

int main(){
    int n, minCount;
    std::cout << "Введите длину квадрата, для которого вы хотите
получить минимальное распределение" << std::endl;
    std::cin >> n;
    if(!(n>=2 && n<=40)){
        std::cout << "Ошибка, введено неверное число." <<
std::endl;
        return 0;
    }
    std::stack <Square> stackSquares;
    int mul = multiplicity(n);
    int limit = 2* M_PI * sqrt(sqrt((mul == 1) ? n : mul)); //Задаем
предельную величину для количества квадратов
    int **squareArray ;

    if(mul!=1){
        squareArray = new int *[mul];
        for(int i = 0; i < mul; i++)
            squareArray[i] = new int[mul]();
        minCount = findMinSqrS(limit, &stackSquares, squareArray,
mul, 0, n / mul);
    }
    else {
        squareArray = new int *[n];
        for(int i = 0; i < n; i++)
            squareArray[i] = new int[n]();
    }
}

```

```

        int half = n - n / 2;
        fillSquare(squareArray, 0, 0, half);
        fillSquare(squareArray, half, 0, half - 1);
        fillSquare(squareArray, 0, half, half - 1);

        minCount = findMinSqrS(limit - 3, &stackSquares,
squareArray, n, 0, 1) + 3;
        stackSquares.push({half + 1, 1, half - 1});
        stackSquares.push({1, half + 1, half - 1});
        stackSquares.push({1, 1, half});
    }
    Square tmp;
    std::cout << "Для квадрата со стороной " << n << std::endl;
    std::cout << "Количество = " << minCount << std::endl;
    std::cout << "Минимальное распределение квадратов:" <<
std::endl;
    for(int j = 0; j < minCount; j++){
        tmp = stackSquares.top();
        std::cout << j + 1 << ") x = " << tmp.x << ", y = " <<
tmp.y << ", длина = " << tmp.length << std::endl;
        stackSquares.pop();
    }
    if(mul!=1){
        deleteArray(squareArray, mul);
    }
    else{
        deleteArray(squareArray, n);
    }
    return 0;
}

```