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

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Построение и анализ алгоритмов»

Тема: Поиск с возвратом

Студентка гр. 7383 _____ Чемова К.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2019

Цель работы.

Ознакомиться с алгоритмом поиска с возвратом, написать программу для квадрирования квадрата с заданной стороной с использованием поиска с возвратом.

Выполнение работы.

В ходе работы был разработан алгоритм, позволяющий разбить заданный квадрат на более мелкие квадратики. Алгоритм проверяет сторону квадрата на четность, кратность 3 и 5, после чего вызывается соответствующая функция (разбивание квадрата происходит по заранее просчитанным координатам). Это делается для того, чтобы уменьшить время выполнения работы программы. Для остальных случаев сначала вставляется наибольший квадрат в левый верхний угол, после – два одинаковых квадрата справа и снизу от него и вызывается алгоритм поиска с возвратом.

Для выполнения поставленной задачи был написан класс `Square`. Данный класс содержит приватные поля для хранения информации о квадрате и его разбиениях, таких как координаты левых верхних углов, сторон квадратов и их количества.

Также `Square` содержит несколько методов:

- `bool empty(int &cur_x, int &cur_y)` – находит место для вставки нового квадрата;
- `void put_square(int n, int x0, int y0, int side)` – вставляет квадрат по заданным координатам и стороне, то есть заполняет выбранную область `n`;
- `void even()` – разбивает четные исходные квадраты;
- `void div_3()` – разбивает исходные квадраты, кратные 3;
- `void div_5()` – разбивает исходные квадраты, кратные 5;
- `void extend_array()` – добавляет память под массивы данных;
- `int find_max(int x0, int y0)` – находит наибольшую длину стороны квадрата, который можно вставить;
- `void del_square(int x0, int y0, int side)` – удаляет квадрат, то есть заполняет квадрат нулями;
- `Square(int size)` – конструктор, инициализирует поля класса;

- `int first()` – вставляет первый квадрат или на тривиальных случаях сразу разбивает квадрат;
- `void second()` – вставляет второй квадрат;
- `void third()` – вставляет третий квадрат;
- `int backtracking(int deep)` – рекурсивно заполняет остальную площадь квадрата;
- `void print_ans(int k)` – печатает ответ;
- `~Square()` – деструктор;

Исходный код программы представлен в приложении Б.

Тестирование

Программа собрана в операционной системе Ubuntu 17.04 с использованием компилятора g++. В других ОС и компиляторах тестирование не проводилось.

Результаты тестирования показали, что поставленная цель выполнена. Результаты тестирования представлены в Приложении А.

Было изучено необходимое количество итераций при некоторых длинах квадрата. Для этого были использованы простые числа. Результаты исследования представлены в табл. 1.

Таблица 1 – Исследование алгоритма

Длина стороны квадрата	Количество итераций
3	3
5	14
7	60
11	950
13	2370
17	16578
19	65461
23	250838
29	2046067

Выводы.

В ходе выполнения данной работы был изучен метод поиска с возвратом. Была написана программа, использующая метод бэктрекинга для разбиения квадрата на минимально возможное число более мелких квадратов. Была определена сложность алгоритма по количеству вызовов функции, осуществляющей поиск с возвратом: сложность алгоритма не превышает 2^n.

ПРИЛОЖЕНИЕ А
ТЕСТОВЫЕ СЛУЧАИ

Длина стороны квадрата	Результат
8	4 1 1 4 5 1 4 1 5 4 5 5 4
27	6 1 1 18 1 19 9 19 1 9 19 10 9 10 19 9 19 19 9
35	8 1 1 21 1 22 14 22 1 14 22 15 14 15 22 7 15 29 7 22 29 7 29 29 7
37	15 1 1 19 20 1 18 1 20 18 19 20 2 19 22 5 19 27 11 20 19 1 21 19 3 24 19 8 30 27 3 30 30 8 32 19 6 32 25 1 32 26 1 33 25 5

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>

class Square{
  int *x; //координаты x
  int *y; //координыты y
  int *w; //стороны кваратов
  int **part;
  int size; //размер исходного квадрата
  int num;
  int numbers;
  bool final;

  bool empty(int &cur_x, int &cur_y); //находит место для вставки
нового квадрата
  void put_square(int n, int x0, int y0, int side);
  void even(); //разбивает четные исходные квадраты
  void div_3(); //разбивает исходные квадраты, кратные 3
  void div_5(); //разбивает исходные квадраты, кратные 5
  void extend_array();
  int find_max(int x0, int y0);
  void del_square(int x0, int y0, int side);

public:
  Square(int size);
  int first(); //вставляет первый квадрат или на тривиальных случаях
сразу разбивает квадрат
  void second();
  void third();
  int backtracking(int deep);
  void print_ans(int k);
  ~Square();
};

Square::Square(int size):size(size){
  num = 0;
  numbers = 25;
  final = false;
  part = new int*[size];
  for (int i=0; i<size; i++){
    part[i] = new int[size];
```

```
    for (int j=0; j<size; j++)
      part[i][j] = 0;
    }
    x = new int[numbers];
    y = new int[numbers];
    w = new int[numbers];
}

void Square::put_square(int n, int x0, int y0, int side){
  for (int i = 0; i < side; i++)
    for (int j = 0; j < side; j++)
      part[x0 + i][y0 + j] = n;

}

int Square::find_max(int x0, int y0){
  int s1 = 0, s2 = 0, s, i=0, j=0;
  while ((x0 + i) < size && part[x0 + i][y0] == 0){
    s1++;
    i++;
  }
  while ((y0 + j) < size && part[x0][y0 + j] == 0){
    s2++;
    j++;
  }
  (s1 < s2) ? (s = s1) : (s = s2);
  return s;
}

bool Square::empty(int &cur_x, int &cur_y){
  int i = w[0] - 1;
  for (; i < size; i++)
    for (int j = 0; j < size; j++)
      if (part[i][j] == 0){
        cur_x = i;
        cur_y = j;
        return true;
      }
  return false;
}

void Square::extend_array(){
     numbers *= 2;
     int *tmp_x = new int[numbers];
```

```
      int *tmp_y = new int[numbers];
      int *tmp_w = new int[numbers];

      for (int i = 0; i < num; i++){
            tmp_x[i] = x[i];
            tmp_y[i] = y[i];
            tmp_w[i] = w[i];
      }

      delete[] x;
      delete[] y;
      delete[] w;

      x = tmp_x;
      y = tmp_y;
      w = tmp_w;
}

int Square::first(){
  x[num] = 0;
  y[num] = 0;
  if (size%2 == 0){
    w[num] = size/2;
    even();
    return 4;
  }
  if (size%3 == 0){
    w[num] = size*2/3;
    div_3();
    return 6;
  }
  if (size%5 == 0){
    w[num] = size*3/5;
    div_5();
    return 8;
  }
  else{
    w[num] = size/2+1;
    put_square(num+1, 0, 0, w[num]);
    num++;
    second();
    third();
    return backtracking(4);
  }
```

```
}

void Square::second(){
  x[num] = w[0];
  y[num] = 0;
  w[num] = find_max(x[num], y[num]);
  put_square(num+1, x[num], y[num], w[num]);
  num++;
}

void Square::third(){
  x[num] = 0;
  y[num] = w[0];
  w[num] = find_max(x[num], y[num]);
  put_square(num+1, x[num], y[num], w[num]);
  num++;
}

void Square::even(){
  x[2] = y[1] = 0;
  x[1] = y[2] = x[3] = y[3] = w[1] = w[2] = w[3] = size/2;
}

void Square::div_3(){
  x[2] = y[1] = 0;
  x[3] = y[4] = w[1] = w[2] = w[3] = w[4] = w[5] = size/3;
  x[1] = x[4] = x[5] = y[2] = y[3] = y[5] = size*2/3;
}

void Square::div_5(){
  x[2] = y[1] = 0;
  w[4] = w[5] = w[6] = w[7] = size/5;
  x[3] = y[4] = y[5] = w[1] = w[2] = w[3] = size*2/5;
  x[1] = x[4] = y[2] = y[3] = y[6] = size*3/5;
  x[5] = x[6] = x[7] = y[7] = size*4/5;
}

void Square::del_square(int x0, int y0, int side){
     for (int i = 0; i < side; i++)
          for (int j = 0; j < side; j++)
               part[x0 + i][y0 + j] = 0;
}

int Square::backtracking(int deep){
```

```
      if (final && (deep > num))
            return deep;

      int cur_x;
      int cur_y;

      if (!empty(cur_x, cur_y)){
            if (!final || (final && deep - 1 < num))
              num = deep - 1;
            final = true;
            return num;
      }

      if (deep >= numbers)
            extend_array();

      int min_ans = size * size;
      for (int cur_w = find_max(cur_x, cur_y); cur_w > 0; cur_w--) {
            put_square(deep, cur_x, cur_y, cur_w);

            int cur_ans = backtracking(deep + 1);
            min_ans = min_ans < cur_ans ? min_ans : cur_ans;
            if (final && cur_ans <= num){
                  x[deep - 1] = cur_x;
                  y[deep - 1] = cur_y;
                  w[deep - 1] = cur_w;
            }

            del_square(cur_x, cur_y, cur_w);
      }

      return min_ans;
}

void Square::print_ans(int k){
  std::cout << k << std::endl;
  for (int i=0; i<k; i++)
    std::cout << x[i]+1 << " " << y[i]+1 << " " << w[i] << std::endl;
}

Square::~Square(){
  delete[] x;
  delete[] y;
```

```
  delete[] w;
  for (int i=0; i<size; i++)
    delete[] part[i];
  delete[] part;
}

int main(){
  int n, k;
  std::cin >> n;
  Square sq(n);
  k = sq.first();
  sq.print_ans(k);
  return 0;
}
```