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

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Построение и анализ алгоритмов»

Тема: Поиск с возвратом

Студентка гр. 7383 _____ Маркова А. В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2019

Содержание

Цель работы

Исследовать и реализовывать задачу о разбиение квадрата, используя алгоритм поиска с возвратом.

Формулировка 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 её, собрав из уже имеющихся обрезков (квадратов). Внутри столешницы не должно быть пустот, обрезки не должны выходить за пределы столешницы и не должны перекрываться. Кроме того, Вова хочет использовать минимальное возможное число обрезков.

Входные данные: размер столешницы N ($2 \leq N \leq 40$).

Выходные данные: число `amount`, задающее минимальное количество обрезков (квадратов), из которых можно построить столешницу (квадрат) заданного размера N. Далее должны идти `amount` строк, каждая из которых должна содержать три целых числа `abscissa`, `ordinate` и `length`, задающие координаты левого верхнего угла и длину стороны соответствующего обрезка.

Реализация задачи

В данной работе используются главная функция `main()` и дополнительные функции.

Был использован следующий класс:

```
class Square{
private:
    int  **coloring; //раскраска
    int  *abscissa;  //массив координат x
    int  *ordinate;  //массив координат y
    int  *length;    //массив длин сторон квадратов
    int  count;      //возможное кол-во квадратов
    int  size;        //размер текущего квадрата
    int  num;       //порядковый номер квадрата
    bool f;          //послдений возможный квадарт
    void insert_square(int x, int y, int n, int side);
    void remove_square(int x, int y, int side);
    bool place_to_insert(int &x, int &y);
    void multiple_of_three(int side);
    int    find_max_size(int x, int y);
    void multiple_of_five(int side);
    void insert_the_second_square();
    void insert_the_third_square();
    void even_square(int side);
    void print_square();
public:
    int    c; //сложность алгоритма
    void output_of_the_result(int amount);
    int  insert_the_first_square();
```

```
    int  backtracking(int deep);
    Square(int size);
    ~Square();
};
```

`void insert_square(int x, int y, int n, int side)` помещает по заданным координатам `x` и `y` левого верхнего угла квадрат со стороной `side` и номером `n`.

`void remove_square(int x, int y, int side)` удаляет квадрат.

`bool place_to_insert(int &x, int &y)` поиск места для вставки нового квадрата, функция ищет самую верхнюю и левую пустую клетку поля.

`void multiple_of_three(int side)` разбиение квадрата, сторона которого кратна трём.

`int find_max_size(int x, int y)` находит максимальное значение стороны квадрата, который возможно поместить на поле.

`void multiple_of_five(int side)` разбиение квадрата, сторона которого кратна пяти.

`void insert_the_second_square()` вставляет второй квадрат на поле так, чтобы его сторона была максимально возможной.

`void insert_the_third_square()` вставляет третий квадрат на поле так, чтобы его сторона была максимально возможной.

`void even_square(int side)` разбиение квадрата, сторона которого кратна двум.

`void print_square()` вспомогательная функция, которая выводит получившийся квадрат.

`void output_of_the_result(int amount)` выводит результат работы программы на консоль.

`int insert_the_first_square()` функция вставки самого первого квадрата, учитывая размер всего поля: если сторона кратна двум, то вставляет квадрат со стороной в 1/2 стороны поля; если она кратна трём, то 2/3; если она кратна пяти, то 3/5; если же сторона не кратна приведенным выше числам, то сторона первого квадрата будет равна половине стороны поля и + 1. Это нужно для оптимизации работы кода, чтобы уменьшить количество вариантов, в использовании `backtracking`.

`int backtracking(int deep)` перебирает всевозможные варианты кадрирования неразбитого участка поля.

`Square(int size) ~Square()` конструктор и деструктор класса соответственно.

Тривиальное разбиение квадратов представлено на рис. 1.

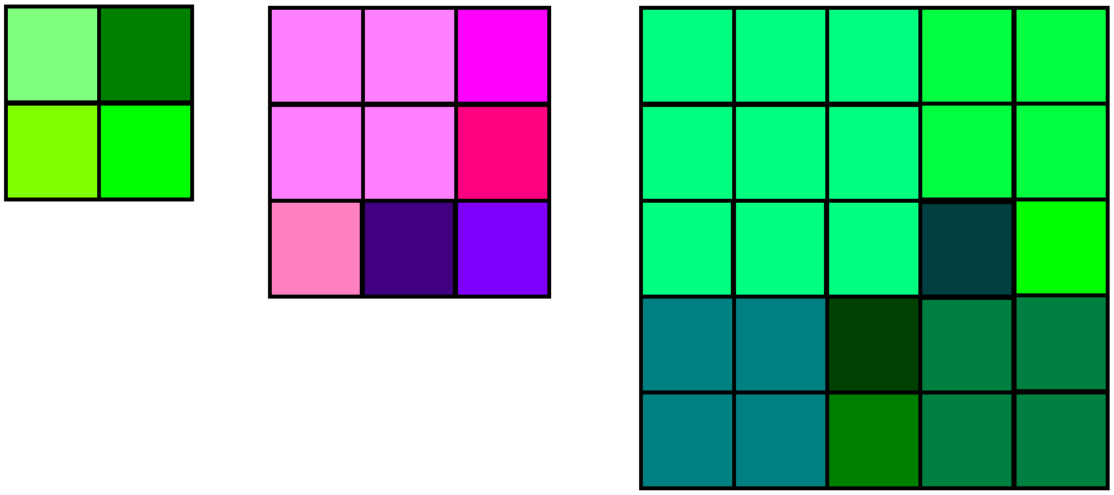

Рисунок 1 – разбиение квадратов, сторона которых кратна 2, 3 и 5

Тестирование

Программа собрана в операционной системе Ubuntu 17.04, с использованием компилятора g++ версии 5.4.0 20160609. В других ОС и компиляторах тестирование не проводилось.

Программа может быть скомпилирована с помощью команды:

g++ – Wall <имя файла>.c

Тестовые случаи представлены в Приложении А.

Исходя из тестовых случаев можно заметить, что тестовые случаи не выявили неправильного поведения программы, что говорит о том, что по результатам тестирование было показано, поставленная задача была выполнена.

Исследование

Для проведения исследования устанавливалась сложность алгоритма по частоте вставки новых квадратов, был добавлен счетчик в функцию `void insert_square(int x, int y, int n, int side)`.

Результаты, полученные в следствии исследования сложности алгоритма, приведены в табл. 1.

Таблица 1 – Результаты исследования

Сторона поля (`size`)	Кол-во вызовов `insert_square`
7	55
11	708
13	1606
17	9964
19	28266
23	105690
29	733269
31	1746940
37	8463490

По этим данным был построен график, который представлен на рис. 2.

Рисунок 2 – зависимость числа операций от вставки квадрата

Отсюда видно, что сложность алгоритма экспоненциальная.

Выводы

В ходе лабораторной работы был изучен алгоритм поиска с возвратом: `backtracking`. Был написан код на языке программирования C++, который применял этот метод для поставленной задачи: кадрирование квадрата.

Так же была исследована сложность алгоритма, которая оказалось экспоненциальная.

ПРИЛОЖЕНИЕ А

Тестовые случаи

Ввод	Вывод	Верно?
4	4 1 1 2 3 1 2 1 3 2 3 3 2	Да
7	9 1 1 4 5 1 3 1 5 3 4 5 2 4 7 1 5 4 1 5 7 1 6 4 2 6 6 2	Да
27	6 1 1 18 19 1 9 19 10 9 19 19 9 1 19 9 10 19 9	Да

ПРИЛОЖЕНИЕ Б

Код программы

```
#include <iostream>
#include <ctime>
#define N 40

class Square{
private:
   int  **coloring; //раскраска
   int  *abscissa;  //массив координат x
   int  *ordinate;  //массив координат y
   int  *length;    //массив длин сторон квадратов
   int  count;      //возможное кол-во квадратов
   int  size;       //размер текущего квадрата
   int  num;        //порядковый номер квадрата
   bool f;          //послдений возможный квадарт

   void insert_square(int x, int y, int n, int side);
   void remove_square(int x, int y, int side);
   bool place_to_insert(int &x, int &y);
   void multiple_of_three(int side);
   int  find_max_size(int x, int y);
   void multiple_of_five(int side);
   void insert_the_second_square();
   void insert_the_third_square();
   void even_square(int side);
   void print_square();

public:
   int  c; //сложность алгоритма
   void output_of_the_result(int amount);
   int  insert_the_first_square();
   int  backtracking(int deep);
    Square(int size);
   ~Square();
};

void Square::insert_square(int x, int y, int n, int side){
//дружественная функция
   c++;
```

```
   for(int i = x; i < side + x; i++)
        for(int j = y; j < side + y; j++)
             coloring[i][j] = n;
}

void Square::remove_square(int x, int y, int side){
   for(int i = x; i < side + x; i++)
        for(int j = y; j < side + y; j++)
             coloring[i][j] = 0;
}

bool Square::place_to_insert(int &x, int &y){
   for(int i = 0; i < size; i++)
        for(int j = 0; j < size; j++)
             if(!coloring[i][j]){
                  x = i;
                  y = j;
                  return true;
             }
   return false;
}

void Square::multiple_of_three(int side){
   abscissa[1] = abscissa[2] = abscissa[3] = ordinate[3] =
ordinate[5] = ordinate[4] = side;
   abscissa[5] = ordinate[2] = side*1/2;
   abscissa[4] = ordinate[1] = 0;
   for(int i = 1; i < 6; i++)
        length[i] = size*1/3;
}

int Square::find_max_size(int x, int y){
   //std::cout<<"I am find_max_size"<<std::endl;
   int  max_size;
   bool allowed = true;
   for(max_size = 1; allowed && max_size <= size - x && max_size <=
size - y; max_size++) //проверка на пересечение границ квадрата
        for(int i = 0; i < max_size; i++)
             for(int j = 0; j < max_size; j++)
                  if(coloring[x + i][y + j]){
                       allowed = false;
                       max_size--;
                  }
```

```
    max_size--;
    return max_size;
}

void Square::multiple_of_five(int side){
    //std::cout<<side<<std::endl;
    abscissa[1] = abscissa[2] = abscissa[7] = ordinate[4] =
ordinate[5] = ordinate[7] = side;
    abscissa[5] = abscissa[6] = ordinate[2] = ordinate[3] = size*2/5;
    length[2] = length[3] = length[5] = length[6] = side*1/3;
    length[1] = length[4] = length[7] = side*2/3;
    abscissa[3] = ordinate[6] = size*4/5;
    abscissa[4] = ordinate[1] = 0;
}

void Square::insert_the_second_square(){
    //std::cout<<"I am insert_the_second_square"<<std::endl;
    abscissa[num] = length[0];
    ordinate[num] = 0;
    length[num] = find_max_size(abscissa[num], ordinate[num]);
    insert_square(abscissa[num], ordinate[num], num + 1,
length[num]);
    num++;
    //print_square();
}

void Square::insert_the_third_square(){
    //std::cout<<"I am insert_the_third_square"<<std::endl;
    abscissa[num] = 0;
    ordinate[num] = length[0];
    length[num] = find_max_size(abscissa[num], ordinate[num]);
    insert_square(abscissa[num], ordinate[num], num + 1,
length[num]);
    num++;
    //print_square();
}

void Square::even_square(int side){
    abscissa[1] = abscissa [3] = ordinate[2] = ordinate[3] =
size*1/2;
    abscissa[2] = ordinate[1] = 0;
    for(int i = 0; i < 4; i++)
         length[i] = size*1/2;
```

```
}

void Square::print_square(){
   for(int i = 0; i < size; i++){
        for(int j = 0; j < size; j++)
             std::cout << coloring[i][j];
        std::cout << std::endl;
   }
   std::cout << std::endl;
}

void Square::output_of_the_result(int amount){
   std::cout << amount << std::endl;
   for(int i = 0; i < amount; i++)
        std::cout << abscissa[i]+1 << " " << ordinate[i]+1 << " " <<
length[i] << std::endl;
   std::cout << c << std::endl;
}

int Square::insert_the_first_square(){
   abscissa[num] = ordinate[num] = 0;
   if(size % 2 == 0){
        //std::cout<<"I am even_square"<<std::endl;
        even_square(size*1/2);
        return 4;
   }
   if(size % 3 == 0){
        //std::cout<<"I am multiple_of_three"<<std::endl;
        length[num] = size*2/3;
        multiple_of_three(length[num]);
        return 6;
   }
   if(size % 5 == 0){
        //std::cout<<"I am multiple_of_five"<<std::endl;
        length[num] = size*3/5;
        multiple_of_five(length[num]);
        return 8;
   }
   else{
        //std::cout<<"I am else"<<std::endl;
        length[num] = size*1/2 + 1;
        insert_square(abscissa[num], ordinate[num], num + 1,
length[num]);
```

```
        num++;
        insert_the_second_square();
        insert_the_third_square();
        return backtracking(4);
    }
}

int Square::backtracking(int deep){
    //std::cout<<"I am backtracking"<<std::endl;
    //print_square();
    if(f && deep > num){
        //std::cout<<deep<<">"<<num<<std::endl;
       //std::cout<<"I am deep"<<std::endl;
        return deep;
    }
    int min_result = size * size;
    int temporary_length;
    int temporary_result;
     int temporary_x;
    int temporary_y;
    if(!place_to_insert(temporary_x, temporary_y)){
        if(!f || (f && deep - 1 < num))
            num = deep - 1;
        f = true;
        return num;
    }
    for(temporary_length = find_max_size(temporary_x, temporary_y);
temporary_length > 0; temporary_length--){
        insert_square(temporary_x, temporary_y, deep,
temporary_length);
        temporary_result = backtracking(deep + 1);
        min_result = min_result < temporary_result ? min_result :
temporary_result;
        if(temporary_result <= num){
            length[deep - 1] = temporary_length;
            abscissa[deep - 1] = temporary_x;
            ordinate[deep - 1] = temporary_y;
        }
        remove_square(temporary_x, temporary_y, temporary_length);
    }
    return min_result;
}
```

```
Square::Square(int size) : size(size){
    coloring = new int*[size];
    for(int i = 0; i < size; i++){
          coloring[i] = new int[size];
          for(int j = 0; j < size; j++)
               coloring[i][j] = 0;
    }
    abscissa = new int[N];
    ordinate = new int[N];
    length = new int[N];
    count = N;
    f = false;
    num = 0;
    c = 0;

}

Square::~Square(){
    delete[] abscissa;
    delete[] ordinate;
    delete[] length;
    for(int i = 0; i < size; i++)
          delete[] coloring[i];
    delete[] coloring;
}

int main(){
    //clock_t time;
    //time = clock();
    int size, count;
    //std::cout << "Size is:" << std::endl;
    std::cin >> size;
    Square table(size);
    count = table.insert_the_first_square();
    table.output_of_the_result(count);
    //time = clock() - time;
    //std::cout<<time/CLOCKS_PER_SEC<<std::endl;
    return 0;
}
```