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

Студент гр. 7383 _____ Медведев И. С.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2019

Содержание

Цель работы

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

Реализация задачи

В данной работе для решения поставленной цели был написан класс Matrix и несколько методов, содержащихся в данном классе.

Конструктор класса создает двумерный массив целых чисел, заполняет его нулями. Так же был написан конструктор копирования.

Метод `void begin` заполняет квадрат тремя стандартными квадратами. Если длина стороны четная, первый квадрат имеет сторону в 1/2 от заданной, если кратна 3, то второй квадрат имеет сторону 2/3 от заданной, если кратна 5 – то 3/5 от исходной длины. Остальные два квадрата ставятся в соседние углы.

Метод `int counter` считает длину стороны вставленного квадрата и возвращает ее.

Метод `bool find_space` возвращает true если находит пустое место, иначе – false.

Метод `bool enough_space` возвращает true если возможно вставить квадрат заданной длины, иначе – false.

Метод `bool insert_square` возвращает true если вставил квадрат заданной длины, иначе – false.

Метод `void clear_board` удаляет квадрат с заданным “цветом”.

Метод `void backtracking` основная рекурсивная функция, которая проверяет какое минимальное количество квадратов поместится в данный квадрат при помощи поиска с возвратом.

Метод `void print_ans` выводит ответ на экран.

Исходный код программы представлен в Приложении А.

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 Б.

Так же было проведено исследование алгоритма. Было изучено необходимое количество итераций при некоторых длинах квадрата. Результаты исследования представлены в табл. 1. В исследовании были использованы простые числа.

Таблица 1 – Исследование алгоритма

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

Из результатов исследования алгоритмов видно, что сложность алгоритма не превышает 2^n, где n – сторона квадрата. Память алгоритм выделяет только для двух двумерных массивов, поэтому по памяти сложность константная.

Вывод

В ходе выполнения лабораторной работы был изучен метод поиска с возвратом, была написана программа для квадрирования квадрата заданной длины и исследован алгоритм.

ПРИЛОЖЕНИЕ А.

КОД ПРОГРАММЫ

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

class Matrix
{
    private:
        int **Matr;
        int m;
        int squares;

        void Create(){
            Matr = new int*[m];
            for (int i = 0; i < m; i++)
                Matr[i] = new int[m];
            for (int i = 0; i < m; i++){
                for (int j = 0; j < m; j++){
                    Matr[i][j] = 0;
                }
             }
        }

    public:
        Matrix(int i): m(i), squares(3){
```

```
    Create();
}

 Matrix(const Matrix& N){
    m = N.m;
    squares = N.squares;
    Matr = new int*[m];
    for (int i = 0; i < m; i++)
        Matr[i] = new int[m];

    for (int i = 0; i < m; i++){
        for (int j = 0; j < m; j++){
            Matr[i][j] = N.Matr[i][j];
        }
    }
}

Matrix &operator = (Matrix const & origin){
    if(this != &origin){
        Matrix temp(origin);
        m = temp.m;
        squares = temp.squares;
        for (int i = 0; i < m; i++){
            for (int j = 0; j < m; j++){
                Matr[i][j] = origin.Matr[i][j];
            }
        }
    }
    return *this;
}
```

```
~Matrix(){
    for (int i = 0; i < m; i++)
        delete[] Matr[i];
    delete[] Matr;
}

int get_squares(){
    return squares;
}


void begin(int k){
    for (int i = m-k; i < m; i++){
        for (int j = m-k; j < m; j++){
            Matr[i][j] = 1;
        }
     }
     for (int i = k; i < m; i++){
        for (int j = 0; j < m-k; j++){
            Matr[i][j] = 2;
        }
     }
     for (int i = 0; i < m-k; i++){
        for (int j = k; j < m; j++){
            Matr[i][j] = 3;
        }
```

```
    }
  }


  void Display(){
    for(int i = 0; i < m; i++){
        for(int j = 0; j < m; j++){
            cout << Matr[i][j];
        }
        cout << endl;
    }
}

  int counter(int x, int y){
      int count = 0;
      int check = Matr[y][x];
      for(int j = x; j < m; j ++){
          if (Matr[y][j] != check)
              return count;
          else
              count++;
      }
      return count;

  }

bool find_space(int& x, int& y){
    for (int i = 0; i < m; ++i){
        for (int j = 0; j < m; ++j){
            if (Matr[i][j] == 0){
```

```
                x = j;
                y = i;
                return true;
            }
        }
    }
    return false;
}

bool enough_space (int x, int y, int size){
    if ((x + size) > m || (y + size) > m)
        return false;
    for (int i = y; i < y + size; ++i){
        for (int j = x; j < x + size; ++j){
            if (Matr[i][j] != 0)
                return false;
        }

    }
    return true;
}



bool insert_square(int x, int y, int size){
    if(!enough_space(x,y,size))
        return false;
    squares++;
    for (int i = 0; i < size; i++)
        for (int j = 0; j < size; j++)
```

```
                Matr[i + y][j + x] = squares;
            return true;
        }

        void clear_board(int x, int y, int size){
            for (int i = y; (i < size) && (Matr[i][x] == squares);
i++)
                for (int j = x; (j < size) && (Matr[i][j] ==
squares); j++)
                    Matr[i][j] = 0;
            squares--;

        }


        void backtracking (Matrix& best, int size,int x, int y, int&
best_numbers){
            if (squares >= best_numbers)
                return;
            for (int i = size; i > 0; --i){
                if (insert_square(x, y, i)){
                    int x1 = x;
                    int y1 = y;
                    if(find_space(x1,y1))
                        backtracking(best, size, x1, y1,
best_numbers);
                    else{
                        if(squares < best_numbers){
                            best_numbers = squares;
                            best = *this;
                        }
```

```
                    clear_board(x, y, size);
                    return;
                }
                clear_board(x, y, size);
            }
        }
    }

    void print_ans(){
        vector<int> check;
        int count = 0;
        for (int i = 0; i < m; i++){
            for(int j = 0; j < m; j++){
                if (find(check.begin(),check.end(), Matr[i][j]) ==
check.end() ){
                    check.push_back(Matr[i][j]);
                    cout<<i+1<<' '<<j+1<<' '<<counter(j,i)<<endl;
                    count = 0;
                }
            }
        }
    }

};
```

```
int main(){
    int size, k;
    cin >> size;
    int best_numbers = size*size;
    if (size % 2 == 0)
        k = size/2;
   else if ( size % 3 == 0)
        k = size*2/3;

    else if (size % 5 == 0)
        k = size*3/5;
    else
        k = (size+1)/2;
    Matrix matr(size);
    Matrix best(size);
    matr.begin(k);
    matr.backtracking(best, (size+1)/2, 0, 0, best_numbers);
    cout<<best_numbers<<endl;
    best.print_ans();
    cout<<endl;
    return 0;
}
```

ПРИЛОЖЕНИЕ Б.

ТЕСТОВЫЕ СЛУЧАИ

Результаты тестирования представлены в табл. 2.

Таблица 2 – Результаты тестирования

Входные данные	Выходные данные
7	9 1 1 2 1 3 2 1 5 3 3 1 2 3 3 1 3 4 1 4 3 1 4 4 4 5 1 3
4	4 1 1 2 1 3 2 3 1 2 3 3 2
9	6 1 1 3 1 4 3 1 7 3 4 1 3 4 4 6 7 1 3
5	8 1 1 2 1 3 1 1 4 2 2 3 1 3 1 1 3 2 1 3 3 3 4 1 2