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. 7304 _____ Овчинников Н.В.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2019

Цель работы

Ознакомиться с алгоритмом поиска с возвратом, получить навыки его программирования и применения на языке C++.

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).

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 $2 \le N \le 20$.

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 ($1 \le x,y \le N$) и длину стороны соответствующего обрезка(квадрата).

Ход работы

1. Создал класс Table, который содержит: размер стола N; двумерный массив размера NxN, в котором ячейка со значение 0 означает пустую клетку, а со значением не равным нулю – занятую квадратом; вектор bestArr объектов класса qdr (класс содержит координаты левого верхнего угла квадрата и ео размер), который хранит квадраты в лучшей расстановке; вектор singleArr объектов класса qdr, который хранит текущие квадраты на столе; bestValue и singleValue – количество квадратов при наилучшей расстановке и текущей соответственно; координаты первой свободной клетки x, y; ключ завершения работы key.
2. Реализовал метод, который в начале программы проверяет делимость размера столешницы на 2 и 3. Если размер делится на 2, то столешница разбивается на 4 квадрата со сторонами N/2. Если размер делится на 3, то столешница разбивается на 6 квадратов: один квадрат со стороной N*2/3, а остальные пять со сторонами N/3.

3. Реализовал рекурсивную метод, который ищет первую свободную клетку на столешнице и ставит в ней квадрат, начиная с самого большого возможного и заканчивая квадратом 1x1. Она вызывается если размер столешницы не делится ни на 2, ни на 3, заранее выставляя на столешнице три квадрата с размерами N/2+1, N/2, N/2. Если методу удалось поставить квадрат, то он вызывается рекурсивно до тех пор, пока весь стол не будет заполнен, либо пока текущее количество квадратов на столе не превысит (или не станет равным) лучшему количеству квадратов в предыдущих расстановках. Когда такое произойдёт, со стола либо будет удалён один последний поставленный квадрат, и на его место встанет квадрат с размером меньше на единицу, либо со стола будут удаляться последние поставленные квадраты, размер которых равен 1. Как только функция удалит квадрат со стороной большей 1, снова вызовется рекурсивная функция с той же позиции. Функция завершит свою работу, когда будут рассмотрены все варианты расстановок квадратов, в каждом из которых количество квадратов не будет превышать лучший, рассмотренный ранее вариант.

Вывод

В ходе выполнения лабораторной работы ознакомился с алгоритмом поиска с возвратом, а также сумел наглядно продемонстрировать его на примере поставленной задачи.

Приложение А

Исходный код программы lr1.cpp

```
#include <iostream>
#include <vector>
#include <cstddef>
#include <stdio.h>

using namespace std;

typedef unsigned char cell;

class qdr
{
public:
    cell x = 0;
    cell y = 0;
    cell w = 0;

public:
    explicit qdr(cell xx = 0, cell yy = 0, cell ww = 0) : x(xx), y(yy), w(ww) {}

    qdr& init(cell xx, cell yy, cell ww)
    {
        x = xx; y = yy; w = ww;
        return *this;
    }
};

class Table
{
private:
    size_t N;
    cell **table;
    vector<qdr> bestArr;
    vector<qdr> singleArr;
    cell bestValue = 255;
    cell singleValue = 0;
    cell x, y;
    size_t max_w;
    bool key = false;

public:
    explicit Table(size_t N)
    {
        if (N > 40 || N < 2)
            N = 0;
        this->N = N;
        max_w = this->N;
        table = new cell*[N];
        for (cell i = 0; i<N; i++)
            table[i] = new cell[N]();
        x = y = 0;
        begin();
    }

    ~Table()
    {
        for (cell i = 0; i<N; i++)
            delete table[i];
        delete table;
    }
```

```
    void getResult()
    {
        go();
        printResult();
    }

private:
    void go()
    {
        if (key)
            return;
        if (!getFirstEmpty() || singleValue >= bestValue)
        {
            if (bestValue > singleValue)
            {
                bestValue = singleValue;
                bestArr = singleArr;
            }

            qdr *tmp = new qdr;
            do
            {
                *tmp = removeQdr();
                if ((*tmp).w == 255)
                {
                    key = true;
                    return;
                }
            } while ((*tmp).w < 2 && !singleArr.empty());

            if (singleArr.empty() && (*tmp).w == 1)
            {
                return;
            }

            (*tmp).w--;
            putQdr(*tmp, 1);
            delete tmp;

            go();
        }
        else
        {
            putQdr();
            go();
        }
    }

    bool getFirstEmpty()
    {
        x = y = 0;
        for (cell i = 0; i<N; i++)
            for (cell k = 0; k<N; k++)
                if (table[i][k] == 0)
                {
                    y = i;
                    x = k;
                    return true;
                }
        return false;
    }

    void putQdr()
    {
        cell max = 0;
```

```
        while (x + max < N && y + max < N && table[y][x + max] == 0 && table[y + max][x] ==
0)
            max++;
        max = max == N ? max - 1 : max;
        max = max > max_w ? max_w : max;
        for (cell i = y; i < y + max; i++)
            for (cell k = x; k < x + max; k++)
                table[i][k] = 1;
        singleArr.push_back(qdr(x, y, max));
        singleValue++;
//printTable();
    }

    void putQdr(qdr &tmp, unsigned char number)
    {
        for (cell i = tmp.y; i < tmp.y + tmp.w; i++)
            for (cell k = tmp.x; k < tmp.x + tmp.w; k++)
                table[i][k] = number;
        singleArr.push_back(tmp);
        singleValue++;
//printTable();
    }

    qdr removeQdr()
    {
        qdr tmp = singleArr.back();
        if (table[tmp.y][tmp.x] == 2)
            return qdr(0, 0, 255);
        for (cell i = tmp.y; i < tmp.y + tmp.w; i++)
            for (cell k = tmp.x; k < tmp.x + tmp.w; k++)
                table[i][k] = 0;
        singleArr.pop_back();
        singleValue--;
//printTable();
        return tmp;
    }

    void begin()
    {
        qdr *tmp = new qdr;
        if (N % 2 == 0)
        {
            putQdr(tmp->init(0, 0, N / 2), 2);
            putQdr(tmp->init(0, N / 2, N / 2), 2);
            putQdr(tmp->init(N / 2, 0, N / 2), 2);
            putQdr(tmp->init(N / 2, N / 2, N / 2), 2);
        }
        else if (N % 3 == 0)
        {
            putQdr(tmp->init(0, 0, N/3*2), 2);
            putQdr(tmp->init(0, N / 3 * 2, N/3), 2);
            putQdr(tmp->init(N / 3, N / 3 * 2, N/3), 2);
            putQdr(tmp->init(N / 3 * 2, 0, N/3) , 2);
            putQdr(tmp->init(N / 3 * 2, N / 3, N / 3), 2);
            putQdr(tmp->init(N / 3 * 2, N / 3 * 2, N / 3), 2);
        }
        else
        {
            putQdr(tmp->init(0, 0, N / 2 + 1), 2);
            putQdr(tmp->init(N / 2 + 1, 0, N / 2), 2);
            putQdr(tmp->init(0, N / 2 + 1, N / 2), 2);
            max_w = N / 2;
        }
        delete tmp;
    }
```

```
    void printResult()
    {
        printf("%u\n", bestValue);
        for (vector<qdr>::iterator it = bestArr.begin(); it != bestArr.end(); it++)
            printf("%u %u %u\n", (*it).x+1, (*it).y+1, (*it).w);
    }

    void printTable()
    {
        for (cell i = 0; i< N; i++)
        {
            for (cell k = 0; k < N; k++)
                printf("%u ", table[i][k]);
            cout << endl;
        }
        cout << endl;
    }
};

int main()
{
    size_t N;
    cin >> N;
    Table answer(N);
    answer.getResult();
    return 0;
}
```