

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

Субботин А.С.

Преподаватель

Филатов А.Ю.

Санкт-Петербург

2019

Цель работы:

Исследование алгоритмов поиска с возвратом, реализация программы заполнения квадрата минимальным числом обрезков.

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 ее, собрав из уже имеющихся обрезков(квадратов).

Например, столешница размера 7×7 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).

Ход работы:

На языке `c++` была написана программа, решающая поставленную задачу методом поиска с возвратом (см. приложение 1), но даже продуманный алгоритм работал непозволительно долго – тогда было решено использовать эвристики для чисел, кратных 2, 3 и 5. Взяты только эти простые числа потому, что на промежутке от 2 до 40 нет квадратов, которые застраивались бы по правилу сторон, кратных семи и более, кроме стороны 7.

Были использованы функции:

`NoHoles` – для выявления незаполненного пространства,

`RewriteSQ` – рекурсивная функция для удаления квадрата со стороной более, чем единица, и замены его на квадрат со стороной на 1 меньше, либо удаление обрезка с размерами 1×1 с рекурсивным вызовом функции.

`SetSQ` – функция, выполняющая поиск свободного места для обрезка и его заполнение.

`FindMAX` – производит оценку того, когда нужно запоминать наилучший вариант, когда необходимо стереть лишние квадраты через `RewriteSQ`, а также вызов `SetSQ`. Завершает работу тогда, когда вектор `square` обнуляется.

`main` – главная функция, в которой производится предварительная оценка задачи и вывод результатов. Вывод выглядит следующим образом:

```
17
12
9 11 3
1 1 9
1 10 8
10 1 8
9 10 1
10 9 2
9 14 4
12 9 4
12 13 1
13 13 5
16 9 2
16 11 2
```

Выводы:

В ходе выполнения данной лабораторной работы был исследован алгоритм поиска с возвратом, а также написана программа, исправно выполняющая поставленные задачи. Проведена работа по оптимизации запрограммированного алгоритма – введено использование эвристик, убрана лишняя часть функционала программы (например, заполнение всего квадрата переменными), проведена замена вектора на сделанный вручную класс. Результат этих действий приведён в приложении 2.

Приложение 1.

```
#include <iostream>
typedef unsigned short int type;
using namespace std;

struct Node{
    type y;
    type x;
    type size;
};

class Mass{
public:
    struct Node* vect;
    type counter;

    Mass() : counter(0){
        vect = new struct Node[80];
    }
    void push_back(struct Node& node){
        vect[counter].size = node.size;
        vect[counter].y = node.y;
        vect[counter++].x = node.x;
    }
    void pop_back(){
        counter--;
    }
    type size(){
        return counter;
    }
    Mass& operator=(Mass& clone){
        counter = clone.size();
        for(type i(0); i < counter; i++){
            vect[i].size = (clone.vect)[i].size;
            vect[i].y = (clone.vect)[i].y;
            vect[i].x = (clone.vect)[i].x;
        }
        return *this;
    }
    bool empty(){
        if(counter)
            return false;
        return true;
    }
    bool Fun(type N){
        if(vect->size < N / 2)
            return false;
        return true;
    }
    struct Node& back(){
        return vect[counter - 1];
    }
    void print(){
        for(type i(0); i < counter; i++){
            cout << vect[i].y+1 << " " << vect[i].x+1 << " " << vect[i].size <<
endl;
        }
    }
};
```

```

class SQ{
public:
    class Mass maxsquare;

    SQ(type size) : maxsize(9999), N(size){
        mass = new type*[N];
        for(type i(0); i < N; i++){
            mass[i] = new type[N];
            for(type j(0); j < N; j++){
                mass[i][j] = 0;
            }
        }

    void FindMAX(){
        while(1){
            if(NoHoles()){
                maxsquare = square;
                maxsize = maxsquare.size();
            }
            else
                SetSQ();
            if(square.size() >= maxsize){
                if(!RewriteSQ())
                    break;
                else if(!square.Fun(N))
                    break;
            }
        }
    }

private:
    type maxsize;
    type** mass;
    type N;
    class Mass square;

    bool NoHoles(){
        type i = 0;
        while(i < N)
            if(mass[N-1][i])
                i += mass[N-1][i];
            else
                return false;
        return true;
    }

    bool RewriteSQ(){
        if(square.empty())
            return false;
        struct Node &node = square.back();

        node.size--;
        mass[node.y + node.size][node.x] = 0;
        if(!node.size){
            square.pop_back();
            return RewriteSQ();
        }
        for(type i(node.y); i < node.y + node.size; i++)
            mass[i][node.x] = node.size;
        return true;
    }
}

```

```

void SetSQ() {
    struct Node node;
    if(square.empty()) {
        node.y = 0;
        node.x = 0;
        node.size = 2*N/3;
    }
    else{
        auto pointer = square.back();
        type i = pointer.y, j = pointer.x;
        type size;
        while(1)
            if(mass[i][j]){
                size = mass[i][j];
                j = (j+size) % N;
                if(j < size)
                    i++;
            }
            else{
                node.y = i;
                node.x = j;
                break;
            }
        node.size = 0;
        for(type it(node.x); it < N; it++)
            if(!mass[node.y][it])
                node.size++;
            else
                break;
        if(node.size > N - node.y)
            node.size = N - node.y;
    }
    for(type i(node.y); i < node.y + node.size; i++)
        mass[i][node.x] = node.size;
    square.push_back(node);
}

};

int main()
{
    type N;
    cin >> N;
    class SQ f(N);
    f.FindMAX();
    cout << f.maxsquare.size() << endl;
    f.maxsquare.print();
    return 0;
}

```

Приложение 2.

```
#include <iostream>
typedef unsigned short int type;
using namespace std;

struct Node{
    type y;
    type x;
    type size;
};

class Mass{
public:
    struct Node* vect;
    type counter;

    Mass() : counter(0){
        vect = new struct Node[80];
    }
    void push_back(struct Node& node){
        vect[counter].size = node.size;
        vect[counter].y = node.y;
        vect[counter++].x = node.x;
    }
    void pop_back(){
        counter--;
    }
    type size(){
        return counter;
    }
    Mass& operator=(Mass& clone){
        counter = clone.size();
        for(type i(0); i < counter; i++){
            vect[i].size = (clone.vect)[i].size;
            vect[i].y = (clone.vect)[i].y;
            vect[i].x = (clone.vect)[i].x;
        }
        return *this;
    }
    bool empty(){
        if(counter)
            return false;
        return true;
    }
    bool Fun(type N){
        if(vect->size < N / 2)
            return false;
        return true;
    }
    struct Node& back(){
        return vect[counter - 1];
    }
    void print(type N){
        cout << counter + 5 << endl;
        cout << "1 1 " << N << endl;
        cout << "1 " << 1+N << " " << N-1 << endl;
        cout << 1+N << " 1 " << N-1 << endl;
        cout << 1+N << " " << N << " 1" << endl;
        cout << N << " " << 1+N << " 2" << endl;
        for(type i(0); i < counter; i++){
            cout << vect[i].y+N << " " << vect[i].x+N << " " << vect[i].size <<
endl;
```

```

    }
}
};

class SQ{
public:
    class Mass maxsquare;

    SQ(type size) : maxsize(9999), N(size){
        mass = new type*[N];
        for(type i(0); i < N; i++){
            mass[i] = new type[N];
            for(type j(0); j < N; j++){
                mass[i][j] = 0;
            }
            mass[0][0] = 3;
            mass[1][0] = 3;
        }

    void FindMAX(){
        while(1){
            if(NoHoles()){
                maxsquare = square;
                maxsize = maxsquare.size();
            }
            else
                SetSQ();
            if(square.size() >= maxsize){
                if(!RewriteSQ())
                    break;
                //else if(!square.Fun(N))
                // break;
            }
        }
    }

private:
    type maxsize;
    type** mass;
    type N;
    class Mass square;

    bool NoHoles(){
        type i = 0;
        while(i < N)
            if(mass[N-1][i])
                i += mass[N-1][i];
            else
                return false;
        return true;
    }

    bool RewriteSQ(){
        if(square.empty())
            return false;
        struct Node &node = square.back();

        node.size--;
        mass[node.y + node.size][node.x] = 0;
        if(!node.size){
            square.pop_back();
            return RewriteSQ();
        }
    }
}

```



```

        for(type i(node.y); i < node.y + node.size; i++)
            mass[i][node.x] = node.size;
        return true;
    }

void SetSQ(){
    struct Node node;
    if(square.empty()){
        node.y = 0;
        node.x = 3;
        node.size = N - 3;
    }
    else{
        auto pointer = square.back();
        type i = pointer.y, j = pointer.x;
        type size;
        while(1)
            if(mass[i][j]){
                size = mass[i][j];
                j = (j+size) % N;
                if(j < size)
                    i++;
            }
            else{
                node.y = i;
                node.x = j;
                break;
            }
        node.size = 0;
        for(type it(node.x); it < N; it++)
            if(!mass[node.y][it])
                node.size++;
            else
                break;
        if(node.size > N - node.y)
            node.size = N - node.y;
    }
    for(type i(node.y); i < node.y + node.size; i++)
        mass[i][node.x] = node.size;
    square.push_back(node);
}

};

int main()
{
    type N;
    cin >> N;
    if(!(N % 2)){
        N /= 2;
        cout << "4" << endl;
        cout << "1 1 " << N << endl;
        cout << 1+N << " 1 " << N << endl;
        cout << "1 " << 1+N << " " << N << endl;
        cout << 1+N << " " << 1+N << " " << N << endl;
        return 0;
    }
    if(!(N % 3)){
        N /= 3;
        cout << "6" << endl;
        cout << "1 1 " << 2*N << endl;
        cout << 1+2*N << " 1 " << N << endl;
        cout << "1 " << 1+2*N << " " << N << endl;
        cout << 1+2*N << " " << 1+N << " " << N << endl;
    }
}

```

```

    cout << 1+N << " " << 1+2*N << " " << N << endl;
    cout << 1+2*N << " " << 1+2*N << " " << N << endl;
    return 0;
}
if(!(N % 5)){
    N /= 5;
    cout << "8" << endl;
    cout << "1 1 " << 3*N << endl;
    cout << 1+3*N << " 1 " << 2*N << endl;
    cout << "1 " << 1+3*N << " " << 2*N << endl;
    cout << 1+3*N << " " << 1+3*N << " " << 2*N << endl;
    cout << 1+2*N << " " << 1+3*N << " " << N << endl;
    cout << 1+2*N << " " << 1+4*N << " " << N << endl;
    cout << 1+3*N << " " << 1+2*N << " " << N << endl;
    cout << 1+4*N << " " << 1+2*N << " " << N << endl;
    return 0;
}
N = N / 2 + 1;
class SQ f(N);
f.FindMAX();
f.maxsquare.print(N);
return 0;
}

```