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. 7304 Юруть Е.А.

Преподаватель Филатов А.Ю.

Санкт-Петербург

2019

Цель работы.

Изучить алгоритм «поиск с возвратом» путем решения задачи квадрирования квадрата.

Основные теоретические положения.

Поиск с возвратом, бэктрекинг — общий метод нахождения решений задачи, в которой требуется полный перебор всех возможных вариантов в некотором множестве M.

Решение задачи методом поиска с возвратом сводится к последовательному расширению частичного решения. Если на очередном шаге такое расширение провести не удается, то возвращаются к более короткому частичному решению и продолжают поиск дальше. Данный алгоритм позволяет найти все решения поставленной задачи, если они существуют. Для ускорения метода стараются вычисления организовать таким образом, чтобы как можно раньше выявлять заведомо неподходящие варианты. Зачастую это позволяет значительно уменьшить время нахождения решения.

Исходный код.

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

using namespace std;

void div_by_2(int N) // Если квадрат кратен двум
{
cout << 4 << endl;
cout << 1 << " " << 1 << " " << N/2 << endl;
cout << 1 << " " << N/2+1 << " " << N/2 << endl;
cout << N/2+1 << " " << 1 << " " << N/2 << endl;
cout << N/2+1 << " " << N/2+1 << " " << N/2 << endl;
}

void div_by_3(int N) // Если квадрат кратен трем
{
cout << 6 << endl;
cout << 1 << " " << 1 << " " << N/3 << endl;
cout << 1 << " " << N/3+1 << " " << N/3 << endl;
cout << 1 << " " << N/3*2+1 << " " << N/3 << endl;
cout << N/3+1 << " " << 1 << " " << N/3 << endl;
cout << N/3*2+1 << " " << 1 << " " << N/3 << endl;
cout << N/3+1 << " " << N/3+1 << " " << N/3*2 << endl;
}

void div_by_5(int N) // Если квадрат кратен пяти
{
cout << 8 << endl;
cout << 1 << " " << 1 << " " << N/5*2 << endl;
cout << 1 << " " << N/5*2+1 << " " << N/5*3 << endl;
cout << N/5*2+1 << " " << 1 << " " << N/5 << endl;
cout << N/5*2+1 << " " << N/5+1 << " " << N/5 << endl;
cout << N/5*3+1 << " " << 1 << " " << N/5*2<< endl;
```

```
cout << N/5*3+1 << " " << N/5*2+1 << " " << N/5 <<
endl;
cout << N/5*4+1 << " " << N/5*2+1 << " " << N/5 <<
endl;
cout << N/5*3+1 << " " << N/5*3+1 << " " << N/5*2 <<
endl;
}

bool free_point(vector< vector<int> > field, int &x,
int &y)
{
    for(int i = 0; i<field.size(); i++)
    {
        for (int j = 0; j<field.size(); j++)
    {
        if(field[i][j] == 0)
        {
            x = j;
            y = i;
            return true;
        }
    }
    }
return false;
}

void field_painting(vector< vector<int> > &field, int
x, int y, int w, int colour) // Закрашивание квадрата
{
for(int i = x; i<x+w; i++)
    {
        for(int j = y; j<y+w; j++)
            field[i][j] = colour;
    }
}
```

```
int max_square(vector< vector<int> > field, int x, int
y)
{
int w_str = 0;
int w_col = 0;

for(int j = x; j<field.size(); j++)
{
    w_str++;

    if(field[y][j] == 1)
        break;
}

for(int i = y; i<field.size(); i++)
    {
    w_col++;

        if(field[i][x] == 1)
        break;
}

if(w_str == w_col) return w_str;
if(w_str > w_col) return w_col;
if(w_str < w_col) return w_str;
}

void backtracking(vector< vector<int> > field,int w,int
N,int x,int y, int &min_size,int colour,int* result)
{
    if(colour >= min_size)
        return;

for(int i = x; i<x+w; i++)
{
    for(int j = y; j<y+w; j++)
```

```
    {
        if(field[i][j] != 0)
            return;
    }
}

colour++;

field_painting(field, x, y, w, colour);

int temp_x = x;
int temp_y = y;


if(free_point(field, y, x) == false)
{
    if(min_size > colour)
    {
            int count = 0;
        int sq_colour = 1;
        int ind = 0;

            for(int i = 0; i<N; i++)
        {
                for(int j = 0; j<N; j++)
            {
                    if(field[i][j] == sq_colour)
                {
                        result[ind++] = j;
                        result[ind++] = i;
                        count = j;

                        while(field[i][count] ==
sq_colour)
                    {
```

```
                            count++;
                        }
                        result[ind++] = count-j;
                        sq_colour++;
                    }
                }
            }
            min_size = colour;
        }
}

else
    free_point(field, y, x);

for(int i = max_square(field, x, y); i>0; i--)

    backtracking(field,i,N,x,y,min_size,colour,result);


    for(int i = temp_y; i<temp_y+w; i++)
        for(int j = temp_x; j<temp_x+w; j++)
            field[i][j] = 0;
}

int main()
{
int N;
cin >> N;

if(N % 2 == 0)
{
    div_by_2(N);
    return 0;
}
```

```
if(N % 3 == 0)
{
    div_by_3(N);
    return 0;
}

if(N % 5 == 0)
{
    div_by_5(N);
    return 0;
}

int min = N;
int* result = new int[N*3];
N = N/2 + 1;

vector< vector<int> > field(N, vector<int>(N, 0));
field[N-1][N-1] = -1;

int x = 0;
int y = 0;
int w = N/2-1;
int colour = 0;

    while(w <= N/2+1)
{
    backtracking(field,w,N,x,y,min,colour,result);
    w++;
}

cout << min+3 << endl;
N = N*2-1;

for(int i = 0; i<min*3; i++)
    {
```

```
    cout << result[i+1]+1 << " " << result[i]+1 << " "
<< result[i+2] << endl;
        i+=2;
    }

cout << N/2+2 << " " << 1 << " " << N/2 << endl;
cout << 1 << " " << N/2+2 << " " << N/2 << endl;
cout << N/2+1 << " " << N/2+1 << " " << N/2+1 << endl;

    return 0;
}
```

Вывод.

В ходе лабораторной работы был реализован алгоритм поиска с возвратом (backtracking) и на его основе была решена задача квадрирования квадрата размером NxN квадратами размера от 1 до N-1.