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 _____ Шарапенков И.И.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2019

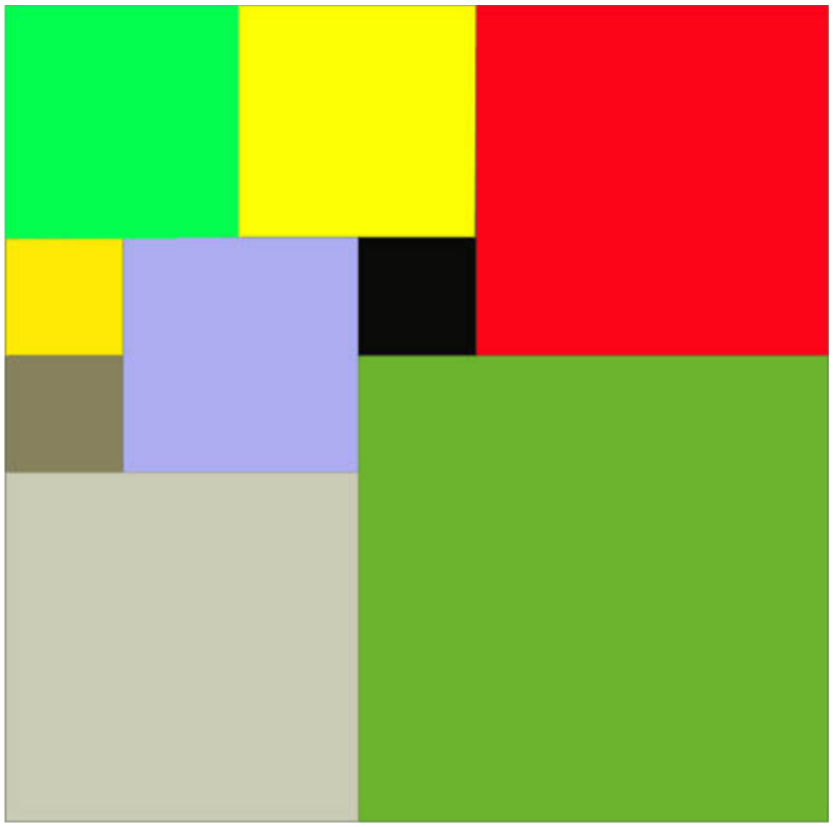
Цель работы.

Изучение алгоритма поиска с возвратом (бэктрекинг).

Задача.

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≤N≤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≤x,y≤N) и длину стороны соответствующего обрезка(квадрата).

Описание алгоритма.

Алгоритм оперирует последовательностями впадин (valley), которые можно визуализировать, если повернуть диаграмму Юнга на 135 градусов против часовой стрелки. После этого пустая область квадрата может быть представлена как последовательность линий, чередующих свое направление вверх и вниз. Каждая пара смежных нисходящих и восходящих линий называется valley (Рис 1).

На каждом уровне поиска выбирается одна из valley и заполняется квадратом. Чтобы избежать дублирования работы, происходит отслеживание квадратов, которые уже были опробованы для данной valley. Для этого в структуре valley хранится нижняя граница размера квадрата, который может быть использован для ее заполнения.

Для усечения дерева поиска также используется простая эвристика:

1. Последовательность из n долин должна быть заполнена по крайней мере n квадратами.
2. Верхняя граница на число квадратов может быть ограничена уравнением (Источник)

$$Bound = 6 * \log_2 N$$

3. Для простого числа p множество минимальных разбиений содержит разбиение в котором в углу стоит квадрат [p/2]+1, а в смежных углах [p/2]

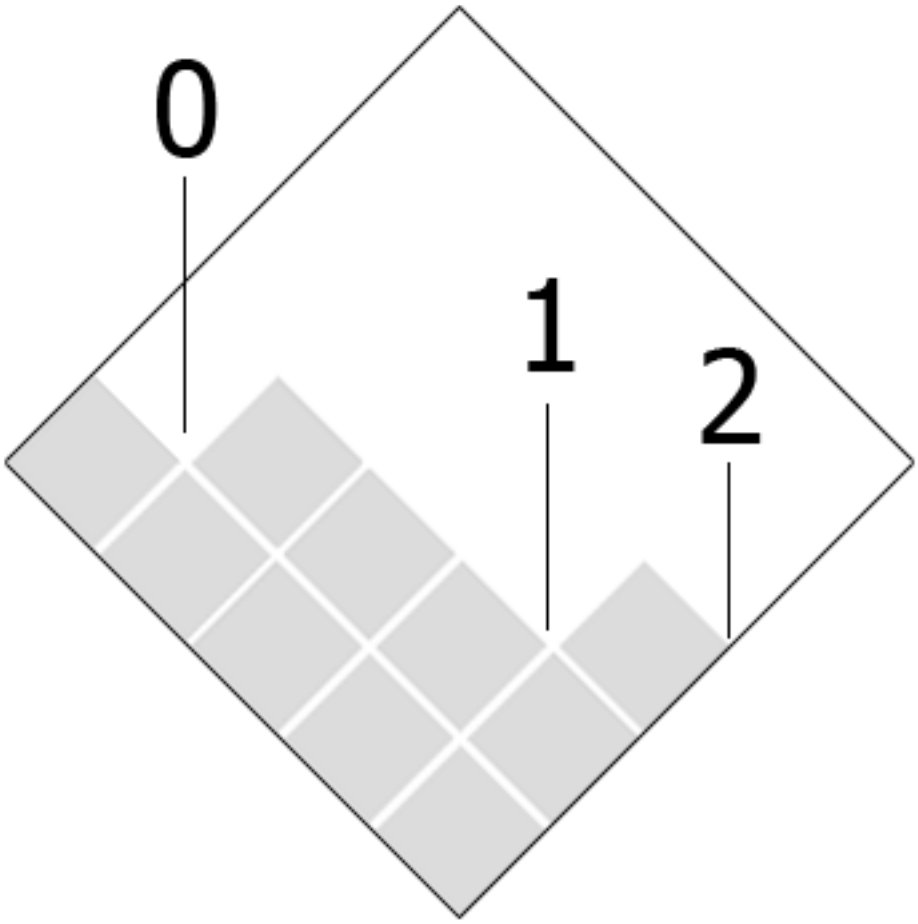


Рис 1. Последовательность valley

Описание структур и функций.

```
// структура для хранения информации о valley
struct valley {
    int l;
    int r;
    int done;
    int x;
    int y;
};

// структура, которая возвращается функциями при нахождении меньшей границы
struct found {
    int level;

    explicit found(int _level)
            : level(_level) {}
};

// двумерный массив для отслеживания состояния заполнения квадрата
static valley valleys[N][N + 1];

// логи для упрощения расшифровки конфигурации
static int logs[N][2];

// основная функция поиска
static void search(int level, int width, int bound);

// оптимизированная функция для случая bound == width
static void searchT(int level, int width)

// оптимизированная функция для случая bound - 1 == width
static void searchT(int level, int width);
```

Результаты

Number: 2
Execution time: 0.001
4
0 0 1
0 1 1
1 0 1
1 1 1

Number: 3
Execution time: 0
6
0 0 2
0 2 1
2 0 1
1 2 1
2 1 1
2 2 1

Number: 4
Execution time: 0.001
4
0 0 2
0 2 2
2 0 2
2 2 2

Number: 5
Execution time: 0
8
0 0 3
0 3 2
3 0 2
2 3 1
2 4 1
3 2 1
4 2 1
3 3 2

Number: 6
Execution time: 0.002
4
0 0 3
0 3 3
3 0 3
3 3 3

Number: 7
Execution time: 0.001
9
0 0 4
0 4 3
4 0 3
3 4 1
4 3 1
4 4 1
3 5 2
5 3 2
5 5 2

Number: 8
Execution time: 0.001
4
0 0 4
0 4 4
4 0 4
4 4 4

Number: 9
Execution time: 0.002
6
0 0 3
0 3 3
0 6 3
3 0 3
6 0 3

Number: 12
Execution time: 0.004
4
0 0 6
0 6 6
6 0 6
6 6 6

Number: 13
Execution time: 0.001
11
0 0 7
0 7 6
7 0 6
6 7 1
7 6 2
6 8 3
6 11 2
9 6 4
9 10 1
8 11 2
10 10 3

Number: 14
Execution time: 0.005
4
0 0 7
0 7 7
7 0 7
7 7 7

Number: 15
Execution time: 0.01
6
0 0 5
0 5 5
0 10 5
5 0 5
10 0 5
5 5 10

Number: 16
Execution time: 0.01
4
0 0 8
0 8 8
8 0 8
8 8 8

Number: 17
Execution time: 0.003
12
0 0 9
0 9 8
9 0 8
8 9 1
9 8 2
8 10 3
11 8 2
11 10 2
11 12 1
8 13 4
13 8 4
12 12 5

Number: 18
Execution time: 0.013
4
0 0 9
0 9 9
9 0 9
9 9 9

Number: 19
Execution time: 0.002

Number: 23
Execution time: 0.004
13
0 0 12
0 12 11
12 0 11
11 12 1
12 11 2
11 13 3
14 11 5
11 16 7
19 11 4
19 15 1
18 16 2
20 15 3
18 18 5

Number: 24
Execution time: 0.073
4
0 0 12
0 12 12
12 0 12
12 12 12

Number: 25
Execution time: 0.281
8
0 0 5
0 5 5
0 10 5
5 0 10
5 10 5
0 15 10
15 0 10
10 10 15

Number: 26
Execution time: 0.18
4
0 0 13
0 13 13
13 0 13
13 13 13

Number: 27
Execution time: 0.333
6
0 0 9
0 9 9
0 18 9
9 0 9
18 0 9
9 9 18

Number: 28
Execution time: 0.175
4
0 0 14
0 14 14
14 0 14
14 14 14

Number: 29
Execution time: 0.01
14
0 0 15
0 15 14
15 0 14
14 15 1
15 14 2
14 16 3
14 19 3
17 14 5
17 19 3

Number: 32
Execution time: 0.425
4
0 0 16
0 16 16
16 0 16
16 16 16

Number: 33
Execution time: 0.729
6
0 0 11
0 11 11
0 22 11
11 0 11
22 0 11
11 11 22

Number: 34
Execution time: 1.438
4
0 0 17
0 17 17
17 0 17
17 17 17

Number: 35
Execution time: 5.842
8
0 0 7
0 7 7
0 14 7
7 0 14
7 14 7
0 21 14
21 0 14
14 14 21

Number: 36
Execution time: 0.684
4
0 0 18
0 18 18
18 0 18
18 18 18

Number: 37
Execution time: 0.017
15
0 0 19
0 19 18
19 0 18
18 19 1
19 18 2
18 20 3
21 18 5
18 23 7
18 30 7
26 18 4
26 22 1
25 23 2
27 22 3
30 18 7
25 25 12

Number: 38
Execution time: 3.526
4
0 0 19
0 19 19
19 0 19
19 19 19

Number: 39
Execution time: 3.159

3 3 6

Number: 10
Execution time: 0.002
4
0 0 5
0 5 5
5 0 5
5 5 5

Number: 11
Execution time: 0.002
11
0 0 6
0 6 5
6 0 5
5 6 1
5 7 1
6 5 1
7 5 1
6 6 2
5 8 3
8 5 3
8 8 3

13
0 0 10
0 10 9
10 0 9
9 10 1
9 11 1
10 9 2
10 11 1
9 12 2
12 9 2
11 11 3
9 14 5
14 9 5
14 14 5

Number: 20
Execution time: 0.019
4
0 0 10
0 10 10
10 0 10
10 10 10

Number: 21
Execution time: 0.095
6
0 0 7
0 7 7
0 14 7
7 0 7
14 0 7
7 7 14

Number: 22
Execution time: 0.056
4
0 0 11
0 11 11
11 0 11
11 11 11

20 19 2
20 21 1
14 22 7
22 14 7
21 21 8

Number: 30
Execution time: 0.262
4
0 0 15
0 15 15
15 0 15
15 15 15

Number: 31
Execution time: 0.007
15
0 0 16
0 16 15
16 0 15
15 16 1
15 17 1
16 15 3
15 18 4
15 22 3
19 15 6
19 21 1
20 21 1
18 22 3
15 25 6
25 15 6
21 21 10

6
0 0 13
0 13 13
0 26 13
13 0 13
26 0 13
13 13 26

Number: 40
Execution time: 1.57
4
0 0 20
0 20 20
20 0 20
20 20 20

Вывод

В ходе лабораторной работы был изучен алгоритм поиска с возвратом (бэктрекинг). Была решена задача о минимальном разбиении квадрата со стороной N на квадраты со сторонами меньше N.

Исходный код программы

```
#include <cstdlib>
#include <iostream>
#include <algorithm>
#include <sstream>
#include <cmath>
#include <time.h>

#define N 41

struct valley {
    int l;
    int r;
    int done;
    int x;
    int y;
};

struct found {
    int level;

    explicit found(int _level)
            : level(_level) {}
};

static valley valleys[N][N + 1];
static int logs[N][2];

static void search(int level, int width, int bound);

static void searchT(int level, int width)
{

    if (width == 0)
        throw found(level);

    for (int i = 0; i < width; i++) {
        if (valleys[level][i].l == valleys[level][i].r &&
            valleys[level][i].done < valleys[level][i].r)
        {
            int w = valleys[level][i].l;
            for (int j = 0; j < i; j++) {
                valleys[level+1][j] = valleys[level][j];
                valleys[level+1][j].done =
                        std::min(valleys[level+1][j].l, valleys[level+1][j].r);
            }
            for (int j = i; j < width-1; j++)
                valleys[level+1][j] = valleys[level][j+1];
            logs[level][0] = i+1;
            logs[level][1] = w;
            if (i > 0)
                valleys[level+1][i-1].r += w;
            valleys[level+1][i].l += w;
            searchT(level+1, width-1);
        }
    }
}

static void searchT1(int level, int width)
{

    if (width == 0)
```

```
        throw found(level);

    for (int i = 0; i < width; i++) {
        int w = std::min(valleys[level][i].l, valleys[level][i].r);
        if (valleys[level][i].done < w) {
            for (int j = 0; j < i; j++) {
                valleys[level+1][j] = valleys[level][j];
                valleys[level+1][j].done =
                        std::min(valleys[level+1][j].l, valleys[level+1][j].r);
            }
            if (valleys[level][i].l == valleys[level][i].r) {
                if (i > 0)
                    valleys[level+1][i-1].r += w;
                valleys[level+1][i].l = valleys[level][i+1].l + w;
                valleys[level+1][i].r = valleys[level][i+1].r;
                valleys[level+1][i].y = valleys[level][i+1].y;
                valleys[level+1][i].x = valleys[level][i+1].x;
                valleys[level+1][i].done = valleys[level][i+1].done;
                for (int j = i+1; j < width-1; j++)
                    valleys[level+1][j] = valleys[level][j+1];
                logs[level][0] = i+1;
                logs[level][1] = w;
                searchT1(level+1, width-1);
            } else if (valleys[level][i].l == w) {
                if (i > 0)
                    valleys[level+1][i-1].r += w;
                valleys[level+1][i].l = valleys[level][i].l;
                valleys[level+1][i].r = valleys[level][i].r - w;
                valleys[level+1][i].y = valleys[level][i].y;
                valleys[level+1][i].x = valleys[level][i].x + w;
                valleys[level+1][i].done = 0;
                for (int j = i+1; j < width; j++)
                    valleys[level+1][j] = valleys[level][j];
                logs[level][0] = i+1;
                logs[level][1] = w;
                searchT(level+1, width);
            } else {
                valleys[level+1][i].l = valleys[level][i].l - w;
                valleys[level+1][i].r = valleys[level][i].r;
                valleys[level+1][i].y = valleys[level][i].y + w;
                valleys[level+1][i].x = valleys[level][i].x;
                valleys[level+1][i].done = 0;
                valleys[level+1][i+1].l = valleys[level][i+1].l + w;
                valleys[level+1][i+1].r = valleys[level][i+1].r;
                valleys[level+1][i+1].y = valleys[level][i+1].y;
                valleys[level+1][i+1].x = valleys[level][i+1].x;
                valleys[level+1][i+1].done = valleys[level][i+1].done;
                for (int j = i+2; j < width; j++)
                    valleys[level+1][j] = valleys[level][j];
                logs[level][0] = i+1;
                logs[level][1] = w;
                searchT(level+1, width);
            }
        }
    }
}

double b;

static void search(int level, int width, int bound) {

    if (width == 0) {
        throw found(level);
```

```
    }

    if (width > bound) {
        return;
    }

    if (width == bound) {
        searchT(level, width);
        return;
    }
    if (width == bound-1) {
        searchT1(level, width);
        return;
    }

    for (int i = 0; i < width; i++) {
        int w = std::min(valleys[level][i].l, valleys[level][i].r);
        int w1 = valleys[level][i].done + 1;
        if (w1 <= w) {
            for (int j = 0; j < i; j++) {
                valleys[level + 1][j] = valleys[level][j];
                valleys[level + 1][j].done =
                        std::min(valleys[level][j].l, valleys[level][j].r);
            }
            if (w1 < w) {
                valleys[level + 1][i].l = valleys[level][i].l - w1;
                valleys[level + 1][i].r = w1;
                valleys[level + 1][i].y = valleys[level][i].y + w1;
                valleys[level + 1][i].x = valleys[level][i].x;
                valleys[level + 1][i].done = 0;
                valleys[level + 1][i + 1].l = w1;
                valleys[level + 1][i + 1].r = valleys[level][i].r - w1;
                valleys[level + 1][i + 1].y = valleys[level][i].y;
                valleys[level + 1][i + 1].x = valleys[level][i].x + w1;
                valleys[level + 1][i + 1].done = 0;
                for (int j = i + 1; j < width; j++)
                    valleys[level + 1][j + 1] = valleys[level][j];
                logs[level][0] = i + 1;
                for (;;) {
                    logs[level][1] = w1;
                    search(level + 1, width + 1, bound - 1);
                    if(++w1 >= w)
                        break;
                    valleys[level + 1][i].l--;
                    valleys[level + 1][i].r = w1;
                    valleys[level + 1][i].y++;
                    valleys[level + 1][i].x = valleys[level][i].x;
                    valleys[level + 1][i + 1].l = w1;
                    valleys[level + 1][i + 1].r--;
                    valleys[level + 1][i + 1].y = valleys[level][i].y;
                    valleys[level + 1][i + 1].x++;
                }
            }
            if (valleys[level][i].l == valleys[level][i].r && level != 0) {
                if (i > 0)
                    valleys[level + 1][i - 1].r += w;
                valleys[level + 1][i].l = valleys[level][i + 1].l + w;
                valleys[level + 1][i].r = valleys[level][i + 1].r;
                valleys[level + 1][i].y = valleys[level][i + 1].y;
                valleys[level + 1][i].x = valleys[level][i + 1].x;
                valleys[level + 1][i].done = valleys[level][i + 1].done;
                for (int j = i + 1; j < width - 1; j++)
                    valleys[level + 1][j] = valleys[level][j + 1];
```

```
                logs[level][0] = i + 1;
                logs[level][1] = w;
                search(level + 1, width - 1, bound - 1);
            } else if (valleys[level][i].l == w) {
                if (i > 0)
                    valleys[level + 1][i - 1].r += w;
                valleys[level + 1][i].l = valleys[level][i].l;
                valleys[level + 1][i].r = valleys[level][i].r - w;
                valleys[level + 1][i].y = valleys[level][i].y;
                valleys[level + 1][i].x = valleys[level][i].x + w;
                valleys[level + 1][i].done = 0;
                for (int j = i + 1; j < width; j++)
                    valleys[level + 1][j] = valleys[level][j];
                logs[level][0] = i + 1;
                logs[level][1] = w;
                search(level + 1, width, bound - 1);
            } else {
                valleys[level + 1][i].l = valleys[level][i].l - w;
                valleys[level + 1][i].r = valleys[level][i].r;
                valleys[level + 1][i].y = valleys[level][i].y + w;
                valleys[level + 1][i].x = valleys[level][i].x;
                valleys[level + 1][i].done = 0;
                valleys[level + 1][i + 1].l = valleys[level][i + 1].l + w;
                valleys[level + 1][i + 1].r = valleys[level][i + 1].r;
                valleys[level + 1][i + 1].y = valleys[level][i + 1].y;
                valleys[level + 1][i + 1].x = valleys[level][i + 1].x;
                valleys[level + 1][i + 1].done = valleys[level][i + 1].done;
                for (int j = i + 2; j < width; j++)
                    valleys[level + 1][j] = valleys[level][j];
                logs[level][0] = i + 1;
                logs[level][1] = w;
                search(level + 1, width, bound - 1);
            }
        }
    }
}

void decryptCoords(int bound, std::stringstream &s) {

    for (int lvl = 0; lvl < bound; lvl++) {
        int order = logs[lvl][0] - 1;
        s << valleys[lvl][order].x << " " << valleys[lvl][order].y << " " <<
logs[lvl][1] << std::endl;
    }

}

bool isPrime(int n) {
    int i, flag = 0;
    if(n == 2) return false;
    for (i = 2; i <= n / 2; ++i) {
        if (n % i == 0) {
            flag = 1;
            break;
        }
    }
    return !flag;
}

int main(int argc, char **argv) {

    int x, y;
```

```
    std::cin >> x;

    y = x;
    std::string reason, coords;

    for(int i = 2; i <= 40; i++) {

        clock_t start = clock();
        x = y = i;

        b = 6*log2(x);

        int bound = int(b);

        if(isPrime(x)) {
            valleys[0][0].l = x;
            valleys[0][0].r = y;
            valleys[0][0].x = 0;
            valleys[0][0].y = 0;
            valleys[0][0].done = x / 2 + 1;
            logs[0][0] = 1;
            logs[0][1] = x / 2 + 1;
            valleys[1][0].l = x / 2;
            valleys[1][0].r = x / 2 + 1;
            valleys[1][0].x = 0;
            valleys[1][0].y = x / 2 + 1;
            valleys[1][0].done = x / 2;
            valleys[1][1].l = x / 2 + 1;
            valleys[1][1].r = x / 2;
            valleys[1][1].x = x / 2 + 1;
            valleys[1][1].y = 0;
            valleys[1][1].done = x / 2;
            logs[1][0] = 1;
            logs[1][1] = x / 2;
            valleys[2][0].l = x / 2;
            valleys[2][0].r = 1;
            valleys[2][0].x = x / 2;
            valleys[2][0].y = x / 2 + 1;
            valleys[2][0].done = 0;
            valleys[2][1].l = x / 2 + 1;
            valleys[2][1].r = x / 2;
            valleys[2][1].x = x / 2 + 1;
            valleys[2][1].y = 0;
            valleys[2][1].done = x / 2;
            valleys[3][0].l = x / 2;
            valleys[3][0].r = 1;
            valleys[3][0].x = x / 2;
            valleys[3][0].y = x / 2 + 1;
            valleys[3][0].done = 0;
            valleys[3][1].l = 1;
            valleys[3][1].r = x / 2;
            valleys[3][1].x = x / 2 + 1;
            valleys[3][1].y = x / 2;
            valleys[3][1].done = 0;
            logs[2][0] = 2;
            logs[2][1] = x / 2;
            for (;;) {
                try {
                    search(3, 2, bound - 1 - 3);
                } catch (found& f) {
                    bound = f.level;
                    std::stringstream s;
                    s << bound << std::endl;
```

```
                    decryptCoords(bound, s);
                    reason = s.str();
                    continue;
                }

                break;
            }
        } else {
            valleys[0][0].l = x;
            valleys[0][0].r = y;
            valleys[0][0].x = 0;
            valleys[0][0].y = 0;
            valleys[0][0].done = 0;
            for (;;) {
                try {
                    search(0, 1, bound - 1);
                } catch (found& f) {
                    bound = f.level;
                    std::stringstream s;
                    s << bound << std::endl;
                    decryptCoords(bound, s);
                    reason = s.str();
                    continue;
                }
                break;
            }
        }

        clock_t stop = clock();

        std::cout << "Number: " << x << std::endl
                << "Execution time: " << double(stop-start) / CLOCKS_PER_SEC <<
std::endl;
        std::cout << reason << std::endl;

    }
}
```