

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. 9382

Преподаватель

Савельев И.С.

Фирсов М.А.

Санкт-Петербург

2021

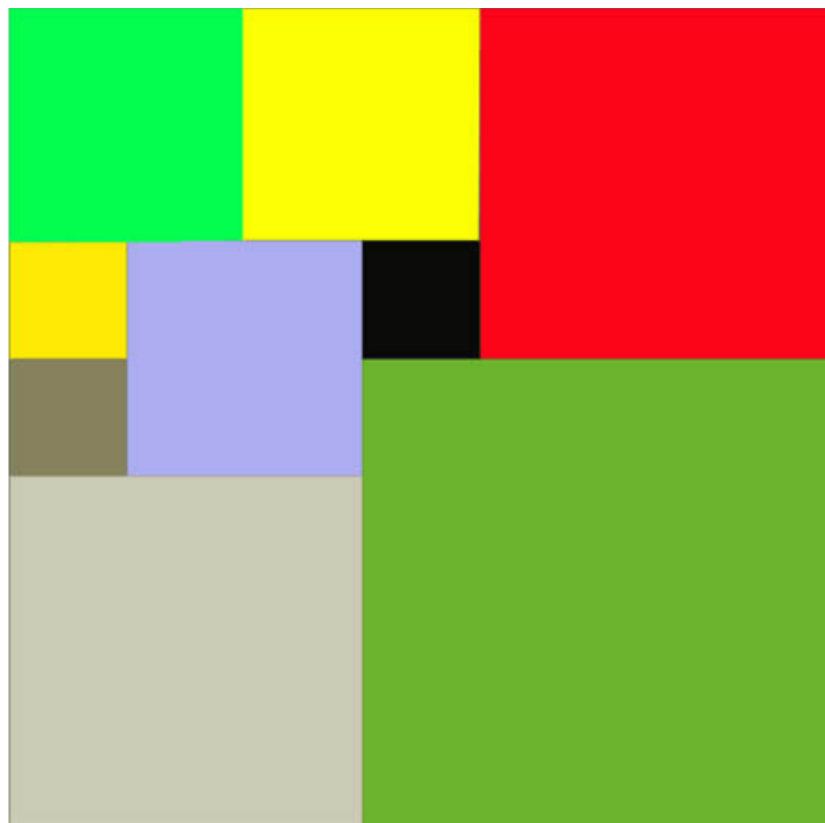
Цель работы.

Познакомиться с основами поиска с возвратом и реализовать соответствующий алгоритм.

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).

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).

Пример входных данных

7

Соответствующие выходные данные

9

1 1 2

1 3 2

3 1 1

4 1 1

3 2 2

5 1 3

4 4 4

1 5 3

3 4 1

Вариант 2р. Рекурсивный бэктрекинг. Исследование времени выполнения от размера квадрата.

Описание алгоритма.

Алгоритм заключается в поиске наименьшего количества квадратов, которыми можно заполнить исходный квадрат со стороной N . Для этого создаются двумерный массива для хранения текущего заполнения квадрата, он сразу же заполняется тремя квадратами которые занимают 75% площади исходного квадрата, также создаются два массива вставляемых квадратов, один

для текущих вставленных квадратов, в него также вносятся три квадрата, а другой для минимального набора вставленных квадратов. В ходе работы алгоритма с помощью рекурсивной функции перебираются все возможные варианты заполнения исходного квадрата. Алгоритм находит пустую область в квадрате и пытается вставить в нее обрезок, размер обрезка определяется с помощью перебора вариантов, начиная с наибольших. Найдя подходящий по размеру обрезок, алгоритм заносит его в двумерный массив и в массив вставленных квадратов. Когда площадь исходного квадрата будет полностью заполнена обрезками алгоритм сравнит текущее количество обрезков с минимальным, если оно меньше то текущее станет минимальным и в массив содержащий наименьший набор квадратов, которыми можно покрыть площадь исходного будет перезаписан информацией из массива содержащего текущие вставленные квадраты. Затем поочередно убираются вставленные квадраты, а на их места алгоритм пробует вставить квадраты других размеров на один меньше чем был вставлен до этого. В ходе перебора до конца не рассматриваются варианты, которые при не полной площади заполнения исходного квадрата имеют количество вставленных квадратов равное минимальному.

Описание функций и структур данных.

`struct Square` структура описывающая вставляемый квадрат, имеет три поля: `int x` - координата по `x`, `int y` - координата по `y`, `int size` - размер вставляемого квадрата

`bool can_insert_sqr(vector<vector<int>>& sqr, int x, int y, int size)` - функция проверяющая возможность вставить квадрат, по координатам `x` и `y`, размера `size`, `sqr` - двумерный массив содержащий текущее заполнение квадрата. Возвращает переменную типа `true`, если можно вставить квадрат размера `size` по координатам `x` и `y`, в противном случае возвращает `false`.

`void insert_sqr(vector<vector<int>>& sqr, int x, int y, int size)` - функция

вставляющая квадрат по координатам x и y , размера $size$, sqr - двумерный массив содержащий текущее заполнение квадрата.

`void print_sqr(vector<vector<int>>& sqr, int k, int N)` - функция выводящая квадрат, sqr - двумерный массив содержащий текущее заполнение квадрата, k - коэффициент сжатия квадрата, N - размер квадрата.

`void init_sqr(vector<vector<int>>& sqr, int N)` - функция инициализирующая начальный квадрат и заполняющая его нулями, sqr - двумерный массив содержащий текущее заполнение квадрат, N - размер квадрата.

`void remove_last_sqr(vector<vector<int>>& sqr, vector<Square>& sqr_mas, int tabb)` - функция удаляющая последний квадрат содержащийся в массиве sqr_mas . sqr_mas - массив содержащий вставленные квадраты, sqr - двумерный массив содержащий текущее заполнение квадрат, $tabb$ - количество отступов для вывода вспомогательной информации в терминал.

`void Start(int &k, vector<vector<int>>& sqr, vector<Square>& sqr_arr, int& S, int& N, int& min_num)` - функция определяющая возможно ли сжать квадрат и инициализирующая начальные переменные. sqr - двумерный массив содержащий текущее заполнение квадрат, S - переменная хранящая текущую не занятую площадь, N - сторона основного квадрата, min_num - переменная хранящая минимальное количество квадратов, которым можно покрыть основной, sqr_arr - массив содержащий вставленные квадраты.

`vector<vector>` - динамический двумерный массив.

Описание рекурсивной функций.

`void recursive_function(vector<vector<int>>& sqr, int cur_space, int cur_size, int sqr_counter, vector<Square>& sqr_arr, int tabb, int N, int& min_num, vector<Square>& min_sqr_arr)` - рекурсивная функция, принимает двумерный массив sqr , cur_space - текущие значение свободной площади в квадрате, cur_size - размер квадрата, который хотим вставить, $sqr_counter$ - количество

квадратов, которые уже вставили, `sqr_arr` - массив содержащий вставленные квадраты, `tabb` - количество отступов для вывода промежуточной информации, `N` - размер первоначального квадрата, `min_num` - минимальное количество квадратов, которыми можно покрыть всю площадь первоначального, `min_sqr_arr` - массив содержащий минимальный набор вставленных квадратов. Сначала функция проверяет, если текущее кол-во обрезков отличается на 1 от минимального и вставляемый квадрат не покрывает всю площадь, то она завершает работу. Затем функция вызывает себя $N/2 - 1$ раз, чтобы обеспечить перебор всех возможных вариантов заполнения квадрата. После чего пробует вставить квадрат размера `cur_size` в свободную область квадрата, если попытка оказалась неудачной то функция завершается, в противном случае происходит проверка, что текущее заполнение является минимальным, если она проходит успешно, то текущее заполнение записывается как минимальное, если не успешно и текущее заполнение уже на данный момент не является минимальным то функция удаляет последний вставленный квадрат и завершается. Если заполнение не является минимальным, но у него есть возможность им стать, то вызывается рекурсивная функция, которая попытается вставить другие квадраты, в конце выполнения функция удаляет вставленный ей квадрат. Функция ничего не возвращает.

Способы хранения частичных решений.

Для хранения квадратов была создана `struct Square`, содержащие координаты левого верхнего угла квадрата, за это отвечают переменные `int x`, `int y` и его размер, за это отвечает переменная `int size`, для хранения главного заполненного квадрата был создан двумерный массив `vector <vector<int>> sqr`. Так же были созданы два массива структур `Square`, один содержит текущие вставленные квадраты `vector <Square> sqr_arr`, а второй содержит минимальный набор вставленных квадратов `vector <Square> min_sqr_arr`.

Оценка сложности по памяти и времени.

В программе используется одна матрица размера N на N , а также другие переменные меньшим образом влияющие на сложность по памяти. Сложность алгоритма по памяти - $O(N^2)$, N - размер исходного квадрата

В исходном квадрате $N \times N$ свободных клеток, количество размеров квадратов которые будут перебираться N . Место для первого квадрата можно выбрать $N^2 \cdot N$ способами. Для второго $(N^2-1) \cdot N$ способами. Таким образом получаем сложность алгоритма по времени равную $O((N^2)! \cdot N^N)$

Использованные оптимизации.

1. Квадрат со стороной кратной 2 заменяется на квадрата со стороной равной 2, в котором и переходит перебор возможных вариантов.
2. Квадрат со стороной кратной 3 заменяется на квадрата со стороной равной 3, в котором и переходит перебор возможных вариантов.
3. Квадрат со стороной кратной 5 заменяется на квадрата со стороной равной 5, в котором и переходит перебор возможных вариантов.
4. Изначально квадрат заполняется на 75%, тремя обрезками размерами $(N+1)/2$, $N/2$, $N/2$, соответственно перебор происходит в оставшейся части.

Исследование.

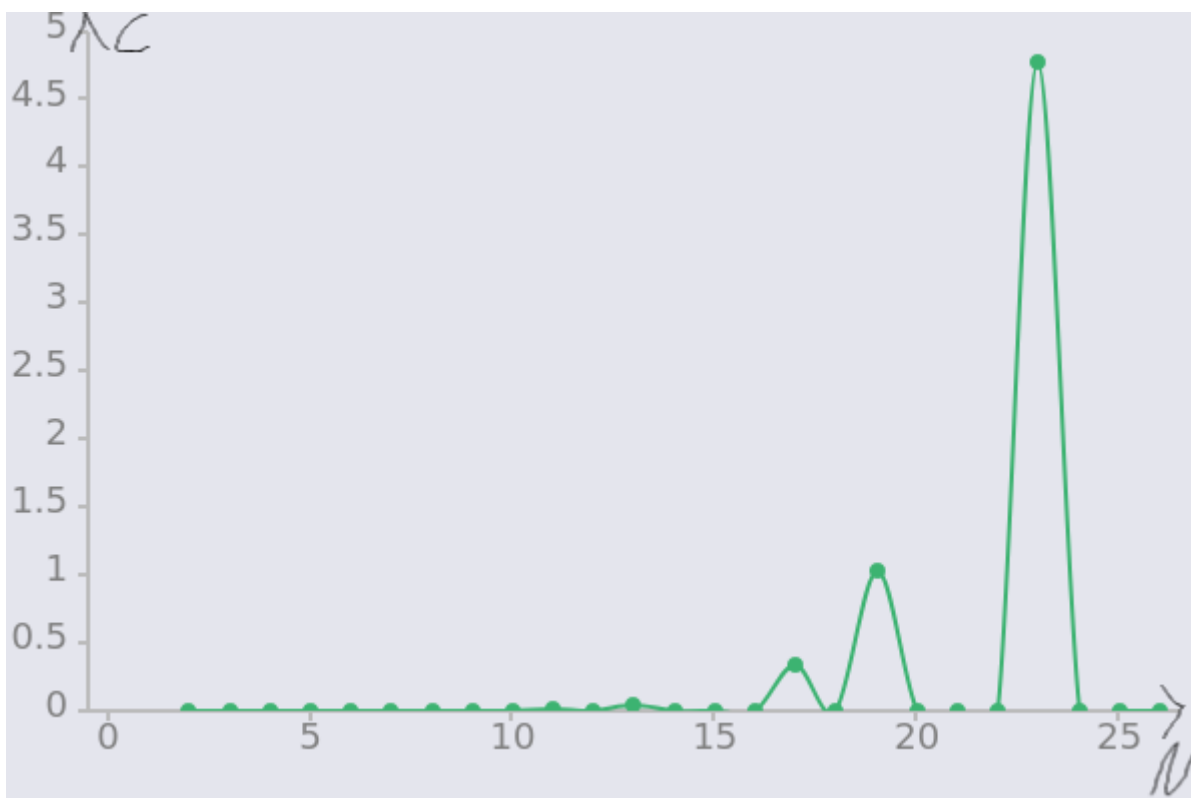
Исследование времени выполнения от размера квадрата. Результаты измерения времени исполнения программы в зависимости от размера квадрата представлены в таблице 1.

Таблица 1.

Сторона квадрата, N	Время выполнения в секундах
2	0.000118
3	0.000331
4	0.000147
5	0.000956

6	0.000151
7	0.002397
8	0.000134
9	0.000234
10	0.000154
11	0.021493
12	0.00013
13	0.045497
14	0.000127
15	0.000268
16	0.000141
17	0.338965
18	0.000124
19	1.03642
20	0.000128
21	0.000265
22	0.000181
23	4.78552
24	0.000125
25	0.00089
26	0.000124

График 1.



На графике можно заметить, что в тех случаях когда не используется оптимизация, то есть сторона квадрата не кратна ни 2, ни 3, ни 5, время выполнения растет экспоненциально.

Тестирование.

Таблица 2 один результаты работы программы

Входные данные	Вывод программы
3	6 1 1 2 1 3 1 3 1 1 3 2 1 2 3 1 3 3 1
5	8 1 1 3 1 4 2 4 1 2

	4 3 2 3 4 1 3 5 1 4 5 1 5 5 1
7	9 1 1 4 1 5 3 5 1 3 5 4 2 7 4 1 4 5 1 7 5 1 4 6 2 6 6 2
8	4 1 1 4 1 5 4 5 1 4 5 5 4
21	6 1 1 14 1 15 7 15 1 7 15 8 7 8 15 7 15 15 7
25	8 1 1 15 1 16 10 16 1 10 16 11 10 11 16 5 11 21 5 16 21 5 21 21 5
31	15 1 1 16 1 17 15 17 1 15

	17 16 3
	20 16 6
	26 16 6
	16 17 1
	16 18 1
	16 19 4
	20 22 1
	21 22 1
	22 22 10
	16 23 6
	16 29 3
	19 29 3

Выводы.

В ходе выполнения данной лабораторной работы была реализована рекурсивная функция реализующая алгоритм поиска с возвратом на языке программирования C++.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ.

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

using namespace std;

// Структура описывающая квадрат
struct Square {
    int x;
    int y;
    int size;
};

// Проверяет возможность вставить квадрат
bool can_insert_sqr(vector<vector<int>>& sqr, int x, int y, int size) {
    // Проверёт, что вставляемый квадрат не выйдет за пределы поля
    if ((x + size) > sqr.size() || (y + size) > sqr.size())
        return 0;
    // Проверяет, что область, в которую вставляют квадрат не занята
    for (int i = y; i < y + size; i++) {
        for (int j = x; j < x + size; j++) {
```

```

        if (sqr[i][j] != 0)
            return 0;
    }
}
return 1;
}

// Вставляет квадрат
void insert_sqr(vector<vector<int>>& sqr, int x, int y, int size) {
    for (int i = y; i < y + size; i++) {
        for (int j = x; j < x + size; j++) {
            sqr[i][j] = size;
        }
    }
}

// Выводит квадрат
void print_sqr(vector<vector<int>>& sqr, int k, int N) {
    for (int i = 0; i < N * k; i++) {
        for (int j = 0; j < N * k; j++) {
            cout.width(3);
            cout << sqr[i][j];
        }
        cout << "\n";
    }
}

// Заполняет начальный массив нулями
void init_sqr(vector<vector<int>>& sqr, int N) {
    sqr.resize(N);
    for (int i = 0; i < N; i++) {
        sqr[i].resize(N);
        for (int j = 0; j < N; j++) {
            sqr[i][j] = 0;
        }
    }
}

// Удаляет последний вставленный квадрат
void remove_last_sqr(vector<vector<int>>& sqr, vector<Square>& sqr_mas, int tabb)
{
    Square tmp = sqr_mas.back();
    sqr_mas.pop_back();
    for(int l=0; l < tabb; l++)
        std::cout << " ";
    std::cout << "Удален " << tmp.size << " (" << tmp.x + 1 << " " << tmp.y + 1
<< ")" << '\n';
    for (int i = tmp.y; i < tmp.y + tmp.size; i++) {
        for (int j = tmp.x; j < tmp.x + tmp.size; j++) {
            sqr[i][j] = 0;
        }
    }
}

void Start(int &k, vector<vector<int>>& sqr, vector<Square>& sqr_arr, int& S, int&
N, int& min_num) {
    init_sqr(sqr, N);
    // Если N кратно 2
    if (N % 2 == 0) {
        k = N / 2;
        N = 2;
    }
}

```

```

// Если N кратно 3
else if (N % 3 == 0) {
    k = N / 3;
    N = 3;
}
// Если N кратно 5
else if (N % 5 == 0) {
    k = N / 5;
    N = 5;
}

min_num = 2 * N + 1;
// Заполняем начальный квадрат
sqr_arr.push_back({ 0, 0, (N + 1) / 2 });
sqr_arr.push_back({ 0, (N + 1) / 2, N / 2 });
sqr_arr.push_back({ (N + 1) / 2, 0, N / 2 });

insert_sqr(sqr, 0, 0, (N + 1) / 2);
insert_sqr(sqr, 0, (N + 1) / 2, N / 2);
insert_sqr(sqr, (N + 1) / 2, 0, N / 2);
std::cout << "Поставил " << (N + 1) / 2 << " (" << 1 << " " << 1 << ")" << '\n';
std::cout << "Поставил " << N / 2 << " (" << 1 << " " << (N + 1) / 2 + 1 <<
")" << '\n';
std::cout << "Поставил " << N / 2 << " (" << (N + 1) / 2 + 1 << " " << 1 <<
")" << '\n';
// Вычисляем площадь после заполнения
S = N * N - ((N + 1) / 2) * ((N + 1) / 2) - 2 * (N / 2) * (N / 2);
}

// Рекурсивная функция
void recursive_function(vector<vector<int>>& sqr, int cur_space, int cur_size, int
sqr_counter, vector<Square>& sqr_arr, int tabb, int N, int& min_num,
vector<Square>& min_sqr_arr) {
    // Если текущее кол-во обрезков отличается на 1 от минимального и
    вставляемый квадрат не покроет всю площадь
    if (sqr_counter == min_num - 1 && cur_space > cur_size*cur_size) {
        for(int l=0; l < tabb; l++)
            std::cout << " ";
        std::cout << "Текущий вариант не минимальный, выход из рекурсии" << '\n';
        return;
    }
    if ((cur_size + 1) <= (N / 2) && sqr_counter == 3) {
        recursive_function(sqr, cur_space, (cur_size+1), sqr_arr.size(), sqr_arr, 0,
        N, min_num, min_sqr_arr);
    }
    // Поиск свободного места и вставка квадрата
    bool flag = false;
    for (int y = 0; y < N; y++) {
        for (int x = 0; x < N; x++) {
            // Если нашли свободное место
            if (sqr[y][x] == 0) {
                // Попытка вставить квадрат размера cur_size
                if (can_insert_sqr(sqr, x, y, cur_size)){
                    insert_sqr(sqr, x, y, cur_size);
                    flag = true;
                    cur_space -= cur_size * cur_size;
                    sqr_arr.push_back({ x, y, cur_size });
                    for(int l=0; l < tabb; l++)
                        std::cout << " ";
                    std::cout << "Поставил " << cur_size<< " (" << x + 1 << " " <<
                    y + 1 << ")" << '\n';

```

```

        break;
    }
    else {
        for(int l=0; l < tabb; l++)
            std::cout << " ";
        std::cout << "Не удалось поставить " << cur_size<< " (" << x +
1 << " " << y + 1 << ")" << '\n';
        return;
    }
    }
    else {
        x += sqr[y][x] - 1;
    }
}
// Выход из внешнего цикла
if(flag) {
    break;
}

// Нашли заполнение равное минимальному на данный момент
if (sqr_counter + 1 == min_num) {
    for(int l=0; l < tabb; l++)
        std::cout << " ";
    std::cout << "Текущий вариант не минимальный, выход из рекурсии" << '\n';
    // Удаляем последний вставленный квадрат
    remove_last_sqr(sqr, sqr_arr, tabb);
    return;
}

// Нашли минимальное заполнение
if (sqr_counter + 1 <= min_num && cur_space == 0) {
    min_num = sqr_counter + 1;
    std::cout << "Найденно новое мин кол-во квадратов: " << min_num << '\n';
    min_sqr_arr.assign(sqr_arr.begin(), sqr_arr.end());
    // Удаляем последний вставленный квадрат
    remove_last_sqr(sqr, sqr_arr, tabb);
    return;
}

// Вызываем рекурсивную функцию
for (int i = N / 2; i > 0; i--) {
    if (i * i <= cur_space) {
        for (int l=0; l < tabb+2; l++)
            std::cout << " ";
        std::cout << "Вызвал рекурсию со стороной квадрата: " << i << '\n';
        recursive_function(sqr, cur_space, i, sqr_counter + 1, sqr_arr,
tabb+2, N, min_num, min_sqr_arr);
    }
}
// Удаляем последний вставленный квадрат
remove_last_sqr(sqr, sqr_arr, tabb);
}

int main() {
    vector <vector<int>> sqr;
    vector <Square> sqr_arr;
    vector <Square> min_sqr_arr;
    int k = 1;
    int S;
    int N;
    int min_num;

```

```

//cout << "Введите число от 2 до 40: ";
cin >> N;

Start(k, sqr, sqr_arr, S, N, min_num);

// Иницилируем переменную хранящую время начала выполнения
clock_t start = clock();
// Вызываем рекурсивную функцию и передаем ей размер квадрата который хотим
вставить (i)

recursive_function(sqr, S, 1, sqr_arr.size(), sqr_arr, 0, N, min_num,
min_sqr_arr);

// Иницилируем переменную хранящую время завершения выполнения
clock_t end = clock();
// Выводим время выполнения
cout << "\nВремя выполнения: " << (double) (end - start) / CLOCKS_PER_SEC <<
"\n\n";

cout << min_num << "\n";

for (int i = 0; i < min_sqr_arr.size(); i++) {
    cout << min_sqr_arr[i].x * k + 1 << " " << min_sqr_arr[i].y * k + 1 << " "
<< min_sqr_arr[i].size * k << "\n";
    insert_sqr(sqr, min_sqr_arr[i].x * k, min_sqr_arr[i].y * k,
min_sqr_arr[i].size * k);
}
std::cout << '\n';
print_sqr(sqr, k, N);
return 0;
}

```