

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
ВАРИАНТ 2Р

Студентка гр. 8303

Потураева М.Ю.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучение алгоритма поиска с возвратом.

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 ,

у 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

Индивидуализация.

Рекурсивный бэктрекинг. Исследование времени выполнения от размера квадрата.

Описание алгоритма.

Работа программы основана на алгоритме поиска с возвратом. Он заключается в полном переборе всех возможных вариантов заполнения квадрата квадратами меньшего размера.

Алгоритм принимает длину стороны столешницы. Далее, в цикле происходит поиск делителя введенного числа и вычисляется коэффициент, который является отношением размеров исходного и заменяющего квадратов. В любом случае мы можем вставить три квадрата с координатами, зависящими от его размера. После вставки запускаем рекурсивный бэктрекинг, в котором после каждой вставки опять рекурсивно вызывается данная функция поиска места и вставки следующего квадрата. Выход из рекурсии происходит тогда, когда на поле не остается пустот.

Для хранения решения используется два вектора: вектор для хранения промежуточных решений (`vector<Square> listSquare`) и итоговый вектор вставленных квадратов (`vector<Square> result`). При возврате отменяется

вставка квадрата находящегося в конце вектора, который является последним вставленным квадратом.

На каждом шаге мы пытаемся вставить квадраты от $N-1$ до 1 .

И максимальная глубина рекурсии будет равна количеству еще не занятых единичных квадратов поля $(N*N + 2N-3)/4$ (при заполнении поля только единичными квадратами). Тогда имеем $(N - 1)^{(N*N + 2N-3)/4}$ вызовов рекурсивной функции и получаем сложность по количеству операций $O(N^{(N*N + 2N-3)/4})$.

На протяжении всей программы используется вектор промежуточных вставок и вектор для хранения результата, двумерный массив указателей для хранения квадрата. Тогда сложность по памяти составляет $O(n*n)$.

Использованные оптимизации.

Два квадрата, размер, одного из которых является наименьшим делимым размером другого, имеют одинаковое минимальное разбиение (с учетом масштаба). Тогда для введенного квадрата ищем такое минимальное делимое, и заменяем размер квадрата. По результату работы программы нужно умножить координаты вставленных квадратов на отношение размеров исходного и заменяющего квадратов.

Если на каком-либо шаге поиска с возвратом в столешнице квадратов больше или равно числу квадратов в лучшем случае, то такой случай пропускается.

Исходная столешница изначально заполняется квадратом со стороной $(N+1)/2$ в левом верхнем углу и квадратами со стороной $(N-1)/2$ в левом нижнем и правом верхнем углах.

Описание структур данных.

Класс Square:

int x, y - координаты квадрата.

int size - размер квадрата.

Void print(int Coeff) – метод вывода результата.

Используется для хранения размещенного на поле квадрата.

vector<Square> listSquare:

Хранит промежуточные решения.

vector<Square> resultSquare:

Вектор для хранения размещенных квадратов.

Описание функций

1) IsCheck_square(int **square, int size, int &x, int &y) – проверка можно ли квадрат размера size установить, начиная с клетки с координатами (x,y).

Функция возвращает true или false, в зависимости от того, можно ли установить данный квадрат.

square-поле для размещения квадратов

size- размер вставляемого квадрата

x- координата x

y-координата y

2) IsFill_square(int **square, int x, int y, int w, int size) - проверка можно ли квадрат размера w существовать, начиная с клетки с координатами (x,y).

Функция возвращает true или false, в зависимости от того, можно ли существовать квадрат в данном участке.

square-поле для размещения квадратов

size- размер участка поля

x- координата x

y-координата y

w- размер вставляемого квадрата

3) insert_square(int **square, int x, int y, int size, int var) - изменение каждой ячейки массива в квадрате с координатой верхнего левого угла (x, y) и размера size на var.

square-поле для размещения квадратов

size- размер вставляемого квадрата

x- координата x

y-координата y

var- значение для заполнения ячеек квадрата

4) delete_square(int **square, int x, int y, int size) - удаления квадрата заданной стороны size, верхний левый угол которого расположен в клетке с координатами (x,y).

square-поле для размещения квадратов

size- размер вставляемого квадрата

x- координата x

y-координата y

5) side_square(int x, int y, int size) - возвращение длины квадрата. Возвращает значение int нового размера квадрата.

size- размер квадрата

x- координата x

y-координата y

6) backtracking(int **square, int size, int &minS, int count) – с помощью алгоритма поиска с возвратом происходит разложение квадрата на минимальное количество меньших квадратов.

square-поле для размещения квадратов

size- размер вставляемого квадрата

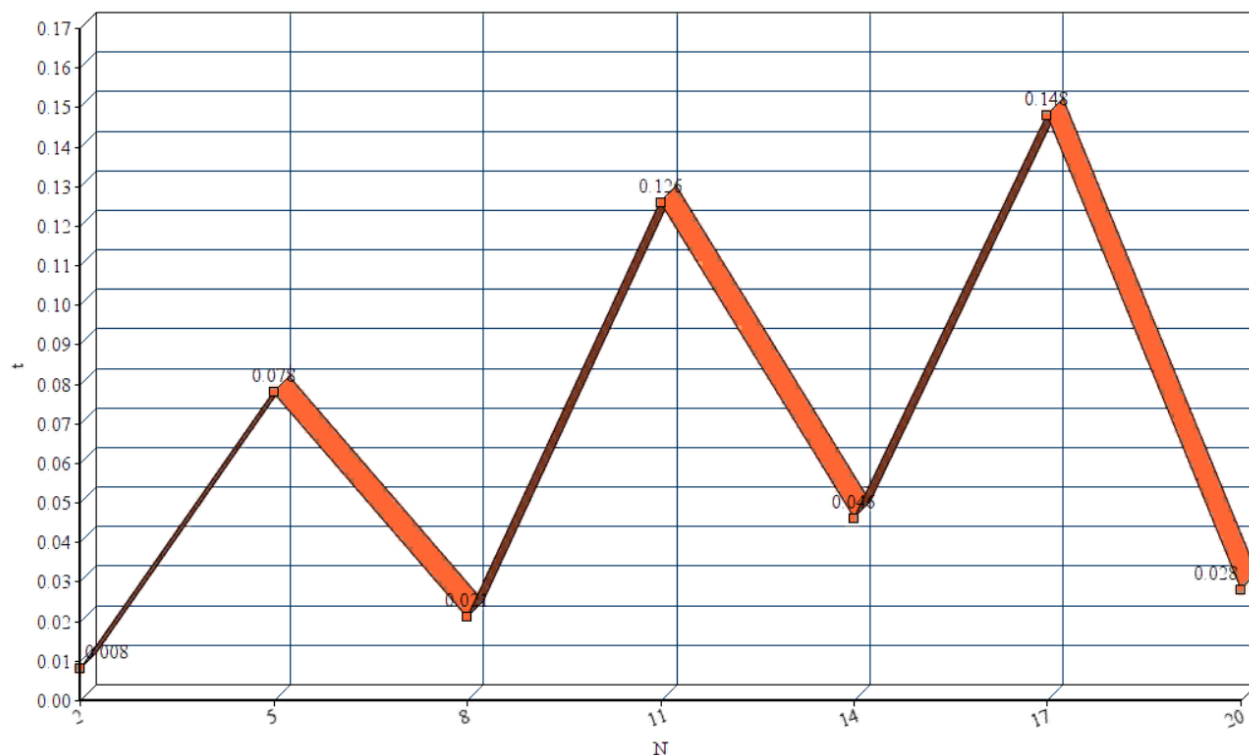
minS-минимальное количество квадратов, которое возможно вставить в поле

count- счетчик количества квадратов

ТЕСТИРОВАНИЕ.

<pre>Enter square size: 9 6 Time:0.048 1 1 6 1 7 3 7 1 3 4 7 3 7 4 3 7 7 3</pre>	<pre>Enter square size: 13 11 Time:0.085 1 1 7 1 8 6 8 1 6 7 8 2 7 10 4 8 7 1 9 7 3 11 10 1 11 11 3 12 7 2 12 9 2</pre>
<pre>Enter square size: 10 4 Time:0.002 1 1 5 1 6 5 6 1 5 6 6 5</pre>	<pre>Enter square size: 19 13 Time:0.063 1 1 10 1 11 9 11 1 9 10 11 3 10 14 6 11 10 1 12 10 1 13 10 4 16 14 1 16 15 1 16 16 4 17 10 3 17 13 3</pre>

Исследование



t,c	0.008	0.078	0.021	0.126	0.046	0.148	0.028
N	2	5	8	11	14	17	20

Исходя из данных, полученных в результате исследования, можно сделать вывод, что время работы программы напрямую зависит от размера квадрата. Также время работы с простыми числами будет больше в несколько раз по сравнению с составными.

Вывод.

В ходе выполнения лабораторной работы был изучен алгоритм поиска с возвратом путем написания программы, решающей задачу по замещению квадратной столешницы заданной длины наименьшим числом квадратов.

ПРИЛОЖЕНИЯ А. ИСХОДНЫЙ КОД

```
#include <iostream>
#include <vector>
#include <ctime>
using namespace std;

//Класс вставленного квадрата в виде координаты-размер квадрата
class Square {
public:
    int x;
    int y;
    int size;
    Square(int x, int y, int size) :x(x), y(y), size(size){}
    void print(int Coeff) {
        cout << x * Coeff + 1 << " " << y * Coeff + 1 << " " << size * Coeff << endl;
    }
};

vector<Square> listSquare;    // промежуточный список квадратов
vector<Square> resultSquare;

bool IsCheck_square(int** square, int size, int& x, int& y) { //проверка можно ли вставить
    квадрат
    for (int i = 0; i < size; i++)
        for (int j = 0; j < size; j++)
            if (square[i][j] == 0) {
                x = i;
                y = j;
                return true;
            }
    return false;
}

bool IsFill_square(int** square, int x, int y, int w, int size) { //проверка может ли суще-
    ствовать квадрат в переданных координатах
    if (x + w > size || y + w > size)
        return false;

    for (int i = x; i < x + w; i++)
        for (int j = y; j < y + w; j++)
            if (square[i][j] != 0)
                return false;

    return true;
}

void insert_square(int** square, int x, int y, int size, int val) { //вставка квадрата
    for (int i = x; i < x + size; i++)
        for (int j = y; j < y + size; j++)
            square[i][j] = val;
}

void delete_square(int** square, int x, int y, int size) { //удаление квадрата
    for (int i = x; i < x + size; i++)
        for (int j = y; j < y + size; j++)
            square[i][j] = 0;
}

int side_square(int x, int y, int size) { //изменение длины квадрата
    if (size - x < size - y)
        return size - x;
```

```

        else
            return size - y;
    }

void backtracking(int** square, int size, int& minSq, int count) {
    if (minSq < count) //если дошли до минимума
        return;

    int x_start = 0, y_start = 0;

    if (IsCheck_square(square, size, x_start, y_start) == false) { //если квадрат можно
вставить
        if (count - 1 < minSq) { //если нашли новый минимум
            minSq = count - 1;
            resultSquare = listSquare;
        }
    }
    else {
        int new_size = side_square(x_start, y_start, size); //иначе возвращаем длину
квадрата
        if (new_size > size - 1) {
            new_size = size - 1;
        }
        for (int i = new_size; i > 0; i--) {
            if (IsFill_square(square, x_start, y_start, i, size)) { //если можем
вставить квадрат
                insert_square(square, x_start, y_start, i, count);
                Square sq(x_start, y_start, i); //создаем новый квадрат
                listSquare.push_back(sq);
                backtracking(square, size, minSq, count + 1); //опять запускаем
бэктрекинг
                listSquare.pop_back(); //возвращаемся
                delete_square(square, x_start, y_start, i); //удаляем квадрат
            }
        }
    }
}

int main() {
    cout << "Enter square size: ";
    int N;
    cin >> N;
    int k = 0;
    int minSq = N + 1; //недостижимое значение количества квадратов
    unsigned int tmp1 = clock();
    for (int i = 2; i <= N; i++) { //уменьшение размера квадрата с помощью нахождения ко-
эффицента
        if (N % i == 0) {
            k = N / i;
            N = i;
            break;
        }
    }

    int** square = new int* [N]; //создаем двумерный массив указателей и инициализируем
его нулями
    for (int i = 0; i < N; i++) {
        square[i] = new int[N];
        for (int j = 0; j < N; j++) {
            square[i][j] = 0;
        }
    }
}

```

```

    int size1 = (N + 1) / 2;
    int size2 = N - size1;
    int count = 1; // в любом случае мы можем изначально вставить три квадрата в соответ-
ствии с его размерами
    insert_square(square, 0, 0, size1, count); count++;
    insert_square(square, 0, size1, size2, count); count++;
    insert_square(square, size1, 0, size2, count); count++;

    listSquare.push_back(Square(0, 0, size1));
    listSquare.push_back(Square(0, size1, size2));
    listSquare.push_back(Square(size1, 0, size2));

    backtracking(square, N-1, minSq, count);
    cout << minSq << endl;
    unsigned int tmp2 = clock();
    cout << (tmp2 - tmp1)/1000.0 << endl;
    for (int i = 0; i < resultSquare.size(); i++) {
        resultSquare[i].print(k); //вывод результата работы программы на экран
    }

    for (int i = 0; i < N; i++)
        delete[] square[i];
    delete[] square;

    return 0;
}

```