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. 7383 _____ Бергалиев М.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург

2019

Цель работы

Реализовать и исследовать алгоритм, основанный на поиске с возвратом.

Постановка задачи

Разбить квадрат размера N на неперекрывающиеся квадраты размеров от 1 до N-1 так, чтобы на нем не было пустот, а количество квадратов было минимальным. Вывести количество квадратов, координаты левых верхних углов квадратов и их размеры.

Ход работы

Была написана программа на языке программирования Python 3. Код представлен в приложении А.

Сначала проверяется размер квадрата n. Для квадрата четного размера количество квадратов равно 4 и возвращается это разбиение. Для квадратов с иным размером находится наименьший простой делитель k его размера. Для квадрата размера k ищется наименьшее разбиение. В этом разбиении присутствуют три квадрата: квадрат размера (k+1)/2 и два квадрата размера k/2. Первый ставится в левый верхний угол, а два других по его бокам и прилегают к сторонам квадрата. Далее остается область размера (k+1)/2 с заполненной верхней левой ячейкой. После находятся два числа m и p такие, что m+p=(k+1)/2, min(|m-p|) и одно из них — простое. К правой стороне области ставятся квадрат размера m и квадрат размера p. Оставшаяся область заполняется бэктрекингом. На каждом шаге в самую верхнюю левую свободную ячейку ставится квадрат наибольшего размера. Когда свободных ячеек не осталось, последний квадрат заменяется квадратом меньшего на 1 размера. Если заменяемый квадрат был размера 1, то он просто удаляется и заменяется предыдущий квадрат. После успешной замены процедура вставки квадратов продолжается. Если на каком-либо шаге вставки количество квадратов превысило лучшее решение, то последний квадрат удаляется и

происходит процедура замены квадратов. Координаты квадратов разбиения квадрата размера k домножаются на n/k. Полученные координаты квадратов и есть разбиение для квадрата размера n.

Тестирование

Тестирование проводилось в Ubuntu 16.04 LTS. По результатам тестирования были выявлены ошибки в коде. Тестовые случаи представлены в приложении Б.

Исследование алгоритма

Исследование проводилось по количеству операций заполнения ячеек для квадратов с размером, являющимся простым. Данные представлены в табл. 1.

Таблица 1 — Результаты исследования зависимости числа операций от размера квадрата

Размер квадрата	Число операций
11	558
13	823
17	3547
19	13664
23	26766
29	233411
31	367551
37	1545674
41	4003103
43	6592819

По полученным данным был построен график, оценивающий сложность алгоритма, показанный на рис. 1. По результатам сложность алгоритма экспоненциальная.

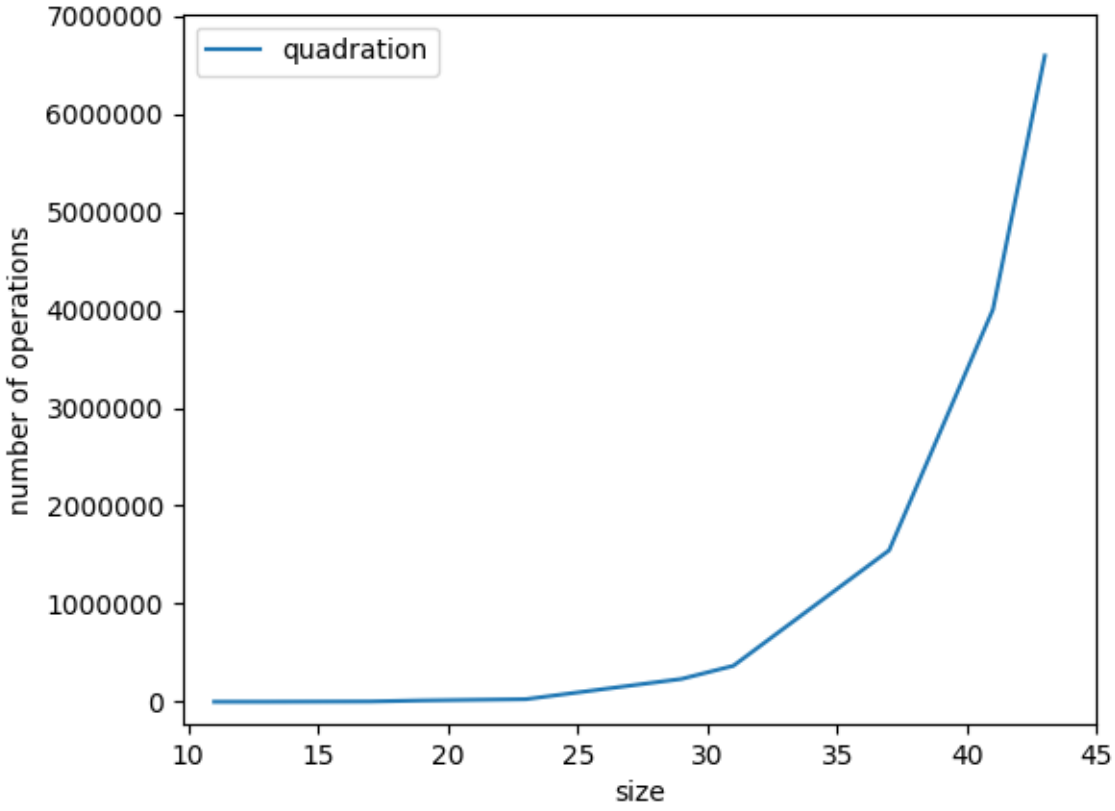

Рисунок 1 — Сложность алгоритма

Вывод

Была изучена идея алгоритмов поиска с возвратом. Был реализован алгоритм поиска минимального квадрирования квадрата с помощью бэктрекинга. Сложность полученного алгоритма — экспоненциальная.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
def fill_square(square, x1, y1, x2, y2, color):
    for i in range(x1, x2):
        for j in range(y1, y2):
            square[i][j] = color


def search_empty(n, square):
    ans = None
    for i in range(0, n):
        for j in range(0, n):
            if square[i][j] == 0:
                ans = [i, j]
                break
        if not ans is None:
            break
    if not ans is None:
        for j in range(0, n-max(ans[0],ans[1])):
            if square[ans[0]+j][ans[1]+j] != 0:
                j -= 1
                break
        for i in range(0, n-ans[1]):
            if square[ans[0]][ans[1]+i] != 0:
                i -= 1
                break
        ans.append(min(j+1, i+1))
        return ans

def quadration(n):
```

```
        if n % 2 == 0:
            return [[0, 0, n//2], [n//2, 0, n//2], [0,
n//2, n//2], [n//2, n//2, n//2]]
        for k in range(3, n+1):
            if n % k == 0:
                break
        ans = _quadration(k)
        c = n // k
        for i in ans:
            i[0] *= c
            i[0] += 1
            i[1] *= c
            i[1] += 1
            i[2] *= c
        return ans


    def _quadration(n):
        square = [[0 for i in range(0, n)] for j in
range(0, n)]
        z = (n+1)//2
        c = n//2
        cur = 1
        fill_square(square, 0, 0, z, z, cur)
        cur += 1
        fill_square(square, 0, z, c, n, cur)
        cur += 1
        fill_square(square, z, 0, n, c, cur)
        cur += 1
        ans = [[0, 0, z], [0, z, c], [z, 0, c]]
        best = []
```

```
        best_count = n*n
        d = 0
        for k in range(2, z+1):
            for i in range(2, k+1):
                if k % i == 0:
                    break
            #if i == k and abs(2*d-z) > abs(2*k-z) and ((k <= (z+1)//2+1 and k >= (z-1)//2-1) or (z-k <=(z+1)//2+1 and z-k >= (z-1)//2-1)):
            if i == k and abs(2*d-z) > abs(2*k-z):
                d = k
        k = d
        if k == z or k == 0:
            return backtraking(square, ans, cur, n*n, n)
        square1 = [j.copy() for j in square]
        ans1 = [j.copy() for j in ans]
        fill_square(square1, n-k, n-k, n, n, cur)
        ans1.append([n-k, n-k, k])
        fill_square(square1, c, n-z+k, c+z-k, n, cur+1)
        ans1.append([c, n-z+k, z-k])
        best1 = backtraking(square1, ans1, cur+2, best_count, n)
        if len(best1) < best_count and len(best1) != 0:
            best = best1
            best_count = len(best1)
        square1 = [j.copy() for j in square]
        ans1 = [j.copy() for j in ans]
        fill_square(square1, n-z+k, n-z+k, n, n, cur)
        ans1.append([n-z+k, n-z+k, z-k])
        fill_square(square1, c, n-k, c+k, n, cur+1)
```

```
        ans1.append([c, n-k, k])
        best1 = backtraking(square1, ans1, cur+2,
best_count, n)
        if len(best1) < best_count and len(best1) != 0:
            best = best1
            best_count = len(best1)
        return best


    def backtraking(square, ans, cur, best_count, n):
        best = []
        end = cur-1
        while cur != end:
            new = search_empty(n, square)
            if new is None:
                if best_count > len(ans):
                    best = [ans[i].copy() for i in
range(0, len(ans))]
                    best_count = len(best)
            while new is None or len(ans) > best_count:
                new = None
                if len(ans) == end:
                    return best
                if ans[-1][2] == 1:
                    square[ans[-1][0]][ans[-1][1]] = 0
                    ans.pop(-1)
                    continue
                for i in range(ans[-1][0], ans[-1]
[0]+ans[-1][2]):
                    square[i][ans[-1][1]+ans[-1][2]-1]
= 0
```

```
                for j in range(ans[-1][1], ans[-1][1]+ans[-1][2]-1):
                    square[ans[-1][0]+ans[-1][2]-1][j] = 0
                ans[-1][2] -= 1
                cur = len(ans) + 1
                break
            if new is None:
                continue
            while square[new[0]+new[2]-1][new[1]+new[2]-1] != 0 or square[new[0]][new[1]+new[2]-1] != 0:
                new[2] -= 1
            fill_square(square, new[0], new[1], new[0]+new[2], new[1]+new[2], cur)
            ans.append(new)
            cur += 1
        return best


    def main():
        n = int(input())
        ans = quadration(n)
        print(len(ans))
        for i in ans:
            print(*i)

    if __name__ == '__main__':
        main()
```

ПРИЛОЖЕНИЕ Б

ТЕСТОВЫЕ СЛУЧАИ

Таблица 1 — Тестовые случаи

Входные данные	Результат
13	11 1 1 7 1 8 6 8 1 6 11 11 3 7 10 4 7 8 2 8 7 1 9 7 3 11 10 1 12 7 2 12 9 2
17	12 1 1 9 1 10 8 10 1 8 13 13 5 9 14 4 9 10 2 9 12 2 10 9 1 11 9 3 11 12 2 13 12 1 14 9 4
29	14 1 1 15 1 16 14 16 1 14 22 22 8 15 23 7 15 16 2 15 18 5 16 15 1 17 15 3 20 15 3 20 18 3 20 21 2

	22 21 1 23 15 7
37	15 1 1 19 1 20 18 20 1 18 30 30 8 19 27 11 19 20 2 19 22 5 20 19 1 21 19 3 24 19 8 30 27 3 32 19 6 32 25 1 32 26 1 33 25 5