МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторная работа №1
по дисциплине «Построение и анализ алгоритмов»
ТЕМА: ПОИСК С ВОЗВРАТОМ

Студент гр. 9303 _____ Павлов Д.Р.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург
2021

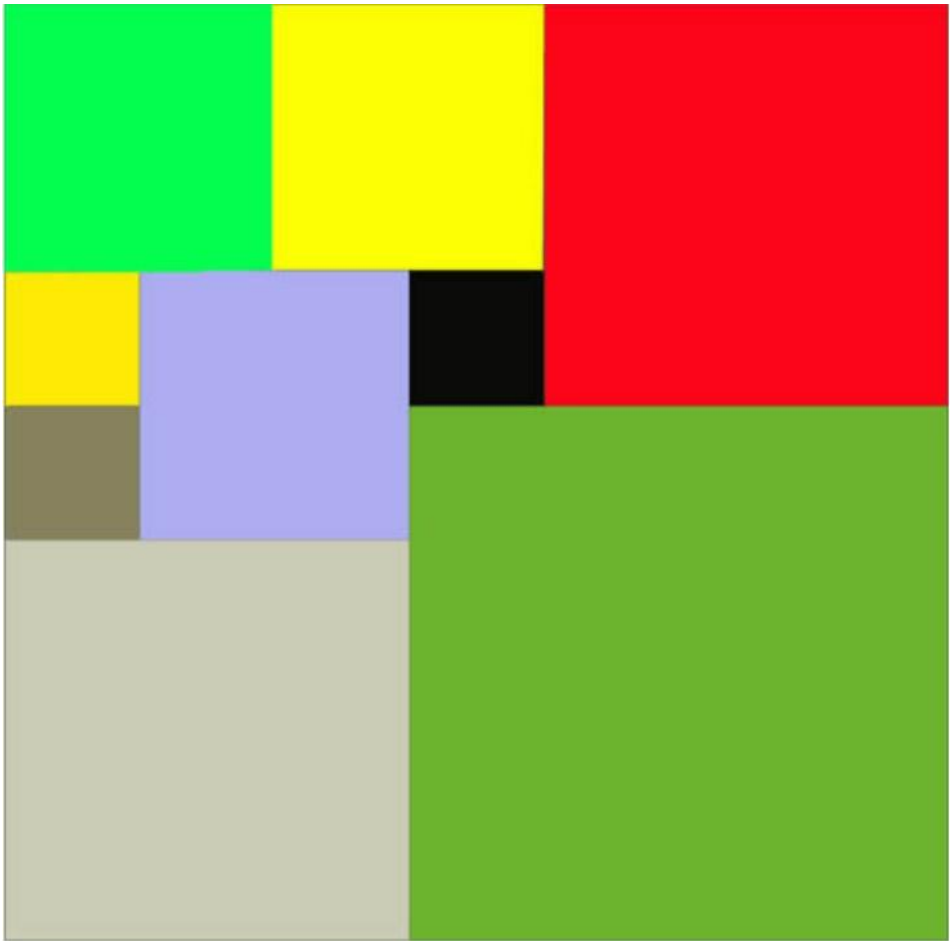
Цель работы.

Изучить метод решения алгоритмических задач «поиск с возвратом» и применить его для решения поставленной задачи.

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.
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 $(2 \le N \le 20)$

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 ($1 \le x, y \le N$) и длину стороны соответствующего обрезка(квадрата).

Вариант 2и. Итеративный бэктрекинг. Исследование времени выполнения от размера квадрата.

Основные теоретические положения.

Поиск с возвратом, бэктрекинг — общий метод нахождения решений задачи, в которой требуется полный перебор всех возможных вариантов в некотором множестве М. Как правило, позволяет решать задачи, в которых ставятся вопросы типа: «Перечислите все возможные варианты ...», «Сколько существует способов ...», «Есть ли способ ...», «Существует ли объект...» и т. п.

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.

Выполнение работы

Описание функций:

`get_sq(side)` –Принимает сторону и возвращает квадрат из нулей.

`set_sq(x, y, w, map, el)` – Устанавливает в координаты квадрат.

`check_sq(x, y, cur, map)` – Проверяет место для квадрата.

`find_sq(map)` – Поиск площади.

`set_for_prime(map, points)` – Возвращает квадрат с установленными квадратами по умолчанию для простых чисел.

`print_points(points)` – Печатает точки.

`find_possible(map, cur)` – Находит варианты.

`code_map(map)` – кодируем карту, увеличивая все значения в ней на 1.

`decode_map(map)` – раскодируем карту, уменьшая все значения списка на 1.

`process(side)` – Принимает сторону и находит самое минимальное кол-во квадратов которое можно установить в исходный квадрат размера side * side.

В таблице ниже приведены размер доски и соответствующее усредненное по 5 испытаниям время работы программы. В данной таблице

приведены размеры квадратов, являющиеся простыми числами, так как понятно, что ответ на задачу для других чисел будет получен быстро.

Таблица 1 – Время работы от размера квадрата.

Размер квадрата	**Время работы, мс.**
3	0.263
5	0.468
7	0.728
11	0.932
13	6.537
17	15.251
19	40.605
23	90.399

На рис. 1 приведен график зависимости времени работы программы в миллисекундах от размера квадрата (n < 30).

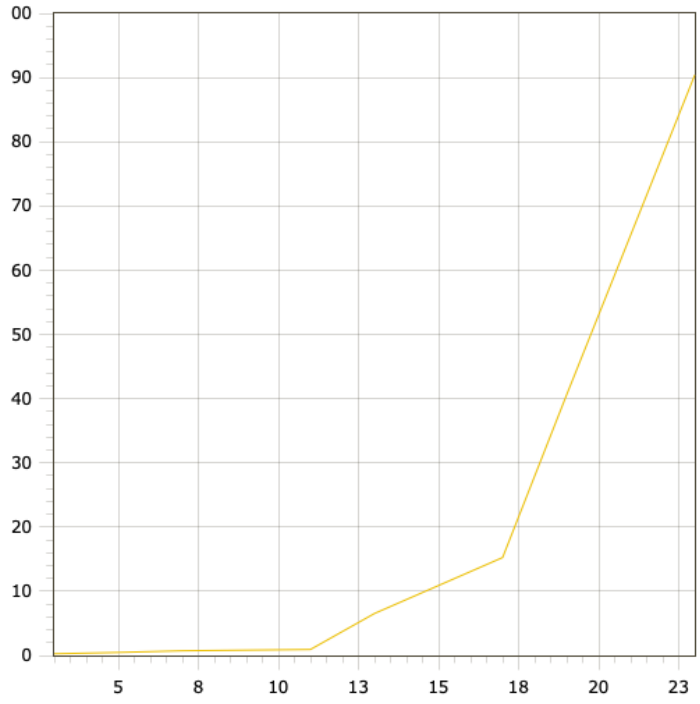


Рисунок 1 – “График зависимости времени от N”.

Из графика можно сделать вывод, что время работы алгоритма от размера входного квадрата зависит экспоненциально.

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Выводы.

Изучил метод решения алгоритмических задач «поиск с возвратом» и применил его для решения поставленной задачи.

Разработана программа, выполняющая считывание с клавиатуры размер доски и в зависимости от свойств данного числа выполняет определенные вычисления, в некоторых случаях применяя «поиск с возвратом», чтобы найти минимальное количество квадратов для замощения и их координаты и длины сторон.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.py

```
# from icecream import ic
# from termcolor import colored
from functions import get_sq, set_sq, print_points, process

if __name__ == '__main__':
    side = int(input('Введите сторону N\n'))
    points = []
    map = get_sq(side)
    count = 0
    if side % 2 == 0:
        # print("Make like cur % 2")
        cur = int(side / 2)
        count = 4
        set_sq(0, 0, side / 2, map, 1)
        set_sq(side / 2, 0, side / 2, map, 2)
        set_sq(0, side / 2, side / 2, map, 3)
        set_sq(side / 2, side / 2, side / 2, map, 4)
        print('4')
        print(0, 0, cur)
        print(0, cur, cur)
        print(cur, 0, cur)
        print(cur, cur, cur)
        exit(0)

    elif side % 3 == 0:
        count += 1
        set_sq(0, 0, side / 3 * 2, map, count)
        points.append([0, 0, int((side / 3) * 2)])
        for i in range(side):
            for j in range(side):
                if map[i][j] == 0:
                    count += 1
                    map = set_sq(j, i, int(side / 3), map, count)
                    points.append([j, i, int(side / 3)])
        print(count)
        print_points(points)
        exit(0)
        # print_map(map)
    else:
        res = process(side)
        best = {'count': res[0], 'map': res[1], 'points': res[2]}

    # print_map(best['map'])
    print(best['count'])
    print_points(best['points'])
```

Файл functions.py

```
from random import randint


def get_sq(side):
    return [[0 for j in range(side)] for i in range(side)]


def set_sq(x, y, w, map, el):
    for line_id in range(len(map)):
        line = map[line_id]
        if line_id < y:
            continue
        if line_id - y > w - 1:
            break
        for sym_id in range(len(line)):
            if sym_id < x:
                continue
            if sym_id - x > w - 1:
                break
            line[sym_id] = el
    return map


def check_sq(x, y, cur, map):
    if (x + cur > len(map)) or (y + cur > len(map)):
        return False
    for i in range(x, x + cur):
        for j in range(y, y + cur):
            if map[j][i] != 0:
                return False
    return True


def find_sq(map):
    sq = 0
    for line in map:
        for s in line:
            if s == 0:
                sq += 1

    return sq


def set_for_prime(map, points):
    side = len(map)
    set_sq(0, 0, (int(side / 2) + 1), map, 1)
    set_sq(int(side / 2) + side % 2, 0, int(side / 2), map, 2)
    set_sq(0, int(side / 2) + side % 2, int(side / 2), map, 3)
    set_sq(int(side / 2) + side % 2, int(side / 2) + side % 2 - 1, 1, map, 4)
    points.append([0, 0, (int(side / 2) + 1)])
    points.append([int(side / 2) + side % 2, 0, int(side / 2)])
    points.append([0, int(side / 2) + side % 2, int(side / 2)])
    points.append([int(side / 2) + side % 2, int(side / 2) + side % 2 - 1,
1])
    return map


def print_points(points):
    # print(points)
    for point in points:
        print(point[0], point[1], point[2])
```

```
def find_possible(map, cur):
    var = []
    for i in range(len(map)):
        for j in range(len(map[i])):
            if map[j][i] == 0:
                if check_sq(i, j, cur, map):
                    var.append([i, j, cur])

    return var


def code_map(map):
    res = []
    for line in map:
        cline = []
        for sym in line:
            cline.append(sym + 1)
        res.append(cline)

    return res


def decode_map(map):
    res = []
    for line in map:
        cline = []
        for sym in line:
            cline.append(sym - 1)
        res.append(cline)

    return res


def process(side):
    points = []
    map = set_for_prime(get_sq(side), points)
    cur = round(0.2684 * side + 0.3398)
    save = code_map(map)
    op = 0
    bestc = round(0.0012 * side ** 3 - 0.0571 * side ** 2 + 1.1161 * side +
3.7250) + 1
    bestm = []
    bestp = []
    svcur = cur + 1
    svpoints = code_map(points)

    while op < 1:
        map = decode_map(save)
        points = decode_map(svpoints)
        count = 4
        cur = svcur - 1
        rest = False
        while find_sq(map) > 0:
            if count > bestc:
                rest = True
                break
            opt = find_possible(map, cur)
            while len(opt) == 0:
                cur -= 1
                opt = find_possible(map, cur)
            if len(opt) == 1:
                opt = opt[0]
```

```
            else:
                opt = opt[randint(0, (len(opt) - 1))]
            count += 1
            map = set_sq(opt[0], opt[1], cur, map, count)
            points.append([opt[0], opt[1], cur])
        if rest:
            continue
        if count < bestc:
            bestc = count
            bestm = map
            bestp = points
            break

return [bestc, bestm, bestp]
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ ПРОГРАММЫ

Таблица Б.1 - Примеры тестовых случаев

No п/п	Входные данные	Выходные данные	Комментарии
1.	5	8 0 0 3 3 0 2 0 3 2 3 2 1 2 3 2 4 3 1 4 4 1 4 2 1	Простое число
2.	10	4 0 0 5 0 5 5 5 0 5 5 5 5	Четное число
3.	15	6 0 0 10 10 0 5 10 5 5 0 10 5 5 10 5 10 10 5	Число, кратное 3

4.	13	11 0 0 7 7 0 6 0 7 6 7 6 1 6 9 4 10 10 3 8 6 3 11 8 2 6 7 2 11 6 2 10 9 1	Простое число