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. 1303 _____ Смирнов Д. Ю.

Преподаватель _____ Фирсов М. А.

Санкт-Петербург
2023

Цель работы.

Изучить принцип работы бэктрекинга, выполнить задание основываясь на этом алгоритме.

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. Представлена на рисунке 1.

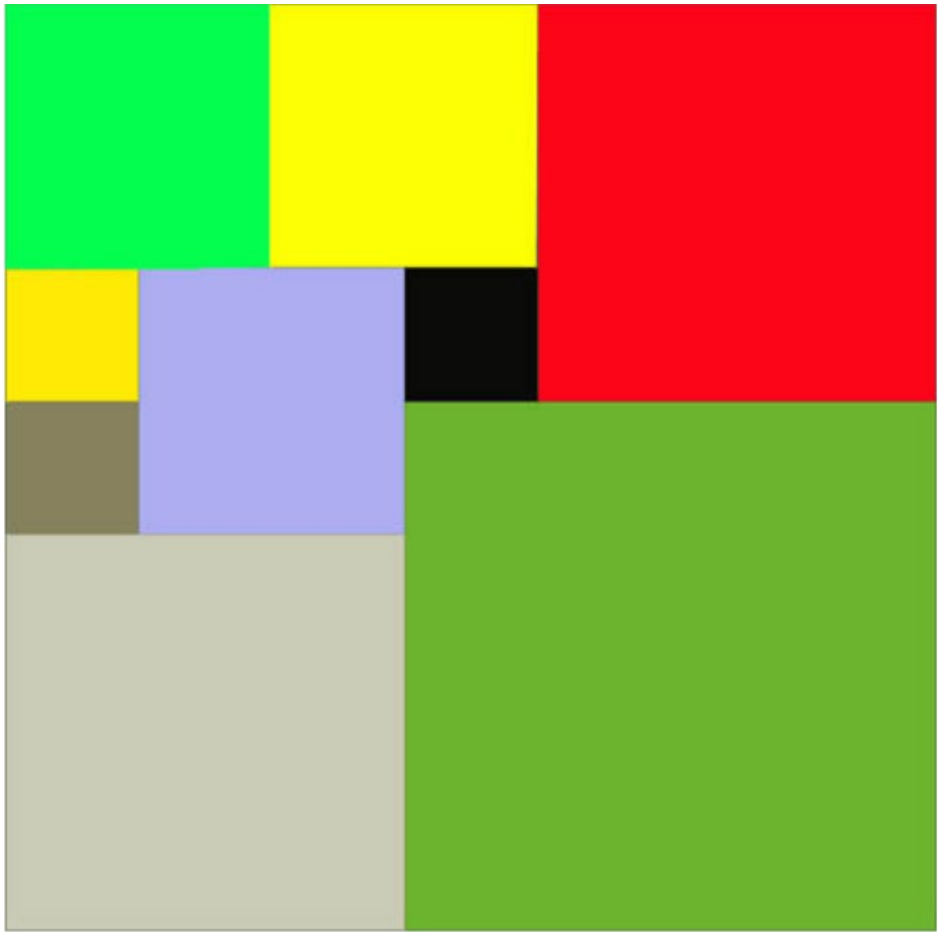

Рисунок 1 - Пример столешницы 7x7

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).

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 **3р**. Рекурсивный бэктрекинг. Исследование кол-ва операций от размера квадрата.

Выполнение работы.

Весь код программы представлен в приложении А.

Оптимизация алгоритма.

1. Будем сохранять количество цветов в минимальной раскраске, то при переборе вариантов можно будет прерывать обход тех ветвей (решений), где количество цветов это значение превосходит.

2. Так как необходимо получать минимальную раскраску, то лучше на каждом шаге сначала ставить квадраты с максимальной стороной и только потом переходить к меньшим квадратам.

3. Каждый раз ставить, а потом стирать целые квадраты – неэффективно. Так как мы решили переходить от больших квадратов к маленьким, то вместо всего квадрата можно стирать лишь некоторую его часть, пример постепенного уменьшения квадрата со стороной 5 представлен на рисунке 2.

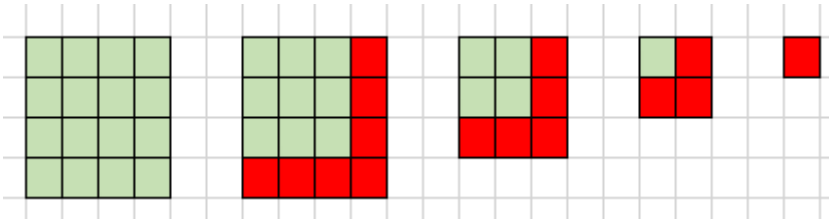

Рисунок 2 - постепенное уменьшение квадрата со стороной 5 (красным обозначены области под удаление)

4. Для четных *N*, минимальное разбиение стола - 4 равных квадрата.

5. Для *N*, кратных простому числу *p*, задачу можно свести к заполнению квадрата со стороной *p*. В конце каждый элемент решения (координата и длина стороны каждого квадрата на столе) на коэффициент *N/p*.

6. Замечено, что во всех раскрасках (для простых *N*) присутствует один большой квадрат размера *(N+1)//2* и два смежных ему квадрата размера *N//2*, пример для квадратов со сторонами 3, 5 и 7 представлен на рисунке 3.

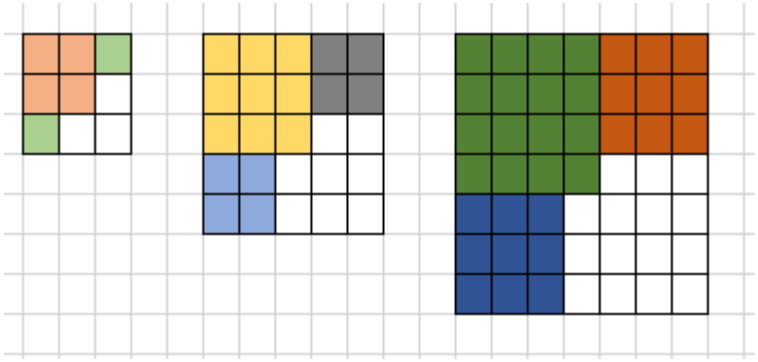

Рисунок 3 - пример вставки первых 3-х квадратов для стола со стороной 3, 5 и 7

Описание алгоритма.

Алгоритм рекурсивного поиска с возвратом для данной задачи реализован следующим образом: находим максимальный размер квадрата который можем вставить и вставляем его. Затем в цикле постепенно перебираем от данного размера до 1го и ищем новую координату для вставки. Если такая есть, то для полученной координаты запускается эта же функция, таким образом развивается это частичное решение. Если вставить нельзя, то проверяем количество цветов в раскраске частичного решения если оно меньше минимальной раскраски, то сохраняем решение и заменяем минимальную раскраску. В конце цикла уменьшаем первый квадрат, который мы вставили.

Таким образом перебираются все возможные варианты разбиений, находится минимальное, заведомо неудачные варианты отбрасываются.

Описание переменных и функций.

Глобальные переменные:

- *DEBUG* – Флаг отвечающий за вывод дополнительной информации.
- *minColor* – Количество цветов в минимальной раскраске.
- *minBoard* – Столешница с минимальной раскраской.
- *minSquares* – Квадраты на столешнице с минимальной раскраской.

- *operationCounter* – Переменная, в которой подсчитывается количество операций, нужна для выполнения условия варианта.

Частичные решения хранятся в виде списка квадратов, участвующих в разбиении стола.

Функции:

- *printBoard(board)* – Функция отвечает за вывод двумерного массива, соответствующего столешнице. Аргументы функции: *board* – двумерный массив, соответствует столешнице, которую хотят вывести на экран. Возвращаемого значения нет.
- *placeSquare(board, xStart, yStart, width, color, squares)* – Функция отвевает за установку квадрата на столешницу, кроме того добавляет квадрат в частичное решение. Аргументы функции: *board* - двумерный массив, соответствует столешнице, на которую ставят квадрат, *xStart* – абсцисса левого верхнего угла квадрата, *yStart* - ордината левого верхнего угла квадрата, *width* – сторона квадрата, *color* – цвет квадрата, *squares* – массив квадратов (частичное решение задачи). Возвращаемого значения нет.
- *sliceSquare(board, xStart, yStart, width, squares)* – Функция отвечает за срез нижней и правой стороны квадрата на полоску с шириной в одну клетку, кроме того уменьшает сторону последнего квадрата в частичном решении на единицу. Аргументы функции: *board* - двумерный массив, соответствует столешнице, на которой уменьшают квадрат, *xStart* – абсцисса левого верхнего угла квадрата, *yStart* - ордината левого верхнего угла квадрата, *width* – сторона квадрата, *squares* – массив квадратов (частичное решение задачи). Возвращаемого значения нет.
- *backTracking(board, xStart, yStart, color, squares)* – Функция, в которой реализован поиск с возвратом. Принцип ее работы описан в разделе «Описание алгоритма». Аргументы функции: *board* - двумерный массив, соответствует столешнице, которую собираются

заполнять, *xStart* – абсцисса левого верхнего угла ещё не заполненной области, *yStart* - ордината левого верхнего угла ещё не заполненной области, *color* – цвет квадрата на текущем шаге, *squares* – массив квадратов (частичное решение задачи). Возвращаемого значения нет.

Оценка сложности алгоритма.

Для теоретической оценки скорости алгоритма, надо проанализировать функцию. Идет перебор по всем координатам, то есть O(N^2) (где N – размер столешницы). Также для каждой координаты рассматриваются возможные размеры квадрата, который можно вставить – выполняется за A операций. Таким образом получаем оценку скорости работы как O(A^{N^2}), где A – некоторая константа.

Для оценки памяти следует рассмотреть худший случай: столешница заполнена единичными квадратами. От сюда следует оценка O(N^2), где N – размер столешницы.

Тестирование.

№ п/п	Входные данные	Выходные данные	Комментарий
1	2	4 1 1 1 2 1 1 1 2 1 2 2 1	Верно.
2	3	6 1 1 2 3 1 1 1 3 1 3 2 1	Верно.

		2 3 1 3 3 1	
3	7	9 1 1 4 5 1 3 1 5 3 5 4 2 7 4 1 4 5 1 7 5 1 4 6 2 6 6 2	Верно.
4	10	4 1 1 5 6 1 5 1 6 5 6 6 5	Верно.
5	17	12 1 1 9 10 1 8 1 10 8 10 9 2 12 9 4 16 9 2 9 10 1 9 11 3 16 11 2 12 13 1 13 13 5	Верно.

		9 14 4	
6	18	4 1 1 9 10 1 9 1 10 9 10 10 9	Верно.
7	20	4 1 1 10 11 1 10 1 11 10 11 11 10	Верно.

Пример вывода дополнительной информации на примере столешницы со стороной 3, представлен на рисунке 4.

```
3
Квадрат со стороной 2 ставят, координаты его левого угла (0:0)
1 1 0
1 1 0
0 0 0
Квадрат со стороной 1 ставят, координаты его левого угла (2:0)
1 1 2
1 1 0
0 0 0
Квадрат со стороной 1 ставят, координаты его левого угла (0:2)
1 1 2
1 1 0
3 0 0
Квадрат со стороной 1 ставят, координаты его левого угла (2:1)
1 1 2
1 1 4
3 0 0
Квадрат со стороной 1 ставят, координаты его левого угла (1:2)
1 1 2
1 1 4
3 5 0
Квадрат со стороной 1 ставят, координаты его левого угла (2:2)
1 1 2
1 1 4
3 5 6
Квадрат со стороной 1 уменьшают, координаты его левого угла (2:2)
1 1 2
1 1 4
3 5 0
Квадрат со стороной 1 уменьшают, координаты его левого угла (1:2)
1 1 2
1 1 4
3 0 0
Квадрат со стороной 1 уменьшают, координаты его левого угла (2:1)
1 1 2
1 1 0
3 0 0
Количество операций: 6
Столецница с минимальным количество квадратов:
1 1 2
1 1 4
3 5 6
Основной ответ на поставленную задачу:
6
1 1 2
3 1 1
1 3 1
3 2 1
2 3 1
3 3 1
```

Рисунок 4 - Вывод дополнительной информации на примере столешницы со стороной 3

Исследование.

Исследование количества операций в зависимости от размера столешницы. За одну операцию примем вставку одного квадрата.

Так как для всех столешниц четного размера единственное решение: 4 квадрата одинакового размера, то исследовать стоит квадраты с нечетной длинной стороны, кроме того, стоит рассматривать столешницы с простой стороной из-за того, что остальные нечетные можно свести к меньшему простому числу. Рассмотрим график, в построении которого участвуют только столешницы со стороной простого числа, представлен на рисунке 5.

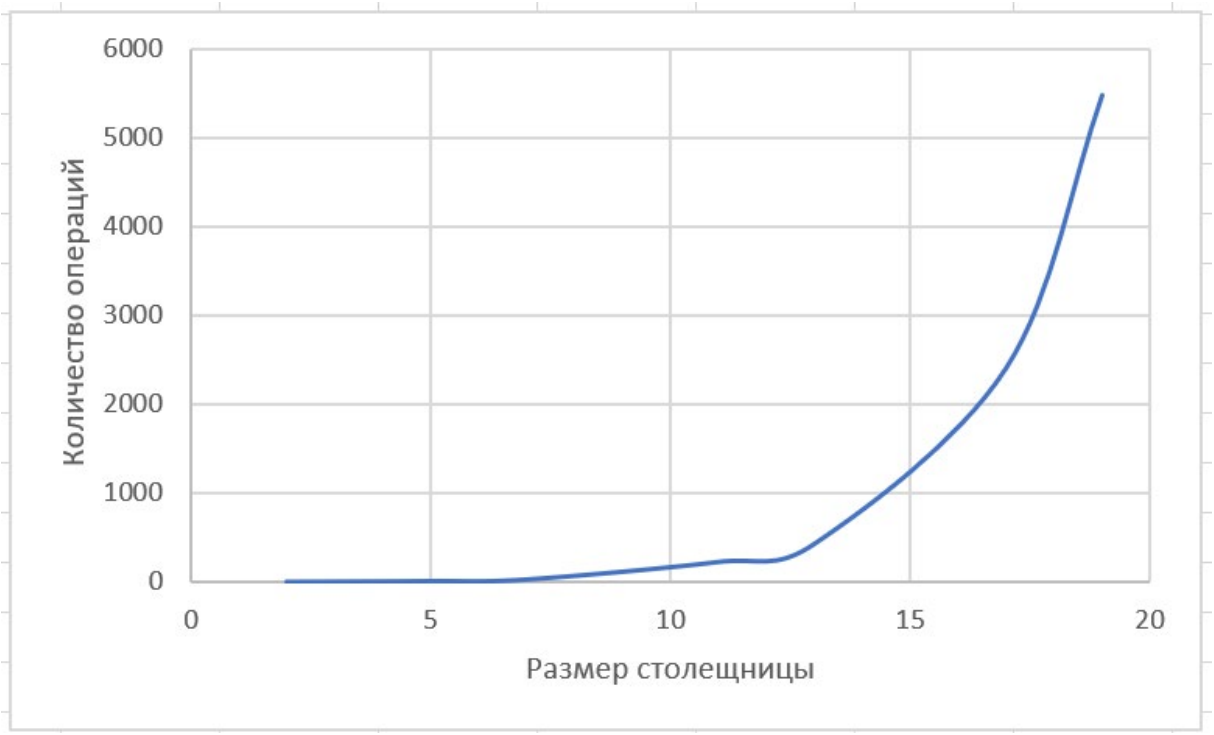

Рисунок 2 - зависимость количества операций от размера столешницы (простые числа)

Данные, по которым строился данный график:

X	2	3	5	7	11	13	17	19
Y	4	6	12	30	227	430	2409	5475

Анализируя данные, видно, что рост количества операций экспоненциальный. То есть сложность алгоритма *backtracking* может быть оценена как O(C^N), где C – константа, N – размер столешницы. Оценка

несколько отличается от аналитической оценки, это происходит из-за того, что в функции применены методы оптимизации, значительно уменьшающие глубину рекурсии, и, как следствие, ресурсозатратность алгоритма.

Вывод.

В результате работы был изучен принцип работы алгоритма поиска с возвратом. С его помощью решена задача. Программа было протестирована, результаты тестов корректны. По результатам исследования выяснено, что время выполнения алгоритма экспоненциально, что подтверждает теоретические данные.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
from copy import deepcopy

DEBUG = False

boardSize = 0
minColor = 0
minBoard = []
minSquares = []
operationCounter = 0


def printBoard(board):
    """
    Функция отвечает за вывод дву мерного массива, соответствующего
столешницу.
    :param board:
    :return: Ничего
    """
    for row in board:
        print(*row)


def placeSquare(board, xStart, yStart, width, color, squares):
    """
    Функция отвевает за установку квадрата на столещницу.
    Функция добавляет квадрат в частичное решение.
    :param board: двумерный массив
    :param xStart: Абсцисса левого верхнего угла квадрата для
вставки
    :param yStart: Ордината левого верхнего угла квадрата для
вставки
    :param width: Сторона вставляемого квадрата
    :param color: Цвет квадрата
    :param squares: Массив частичных решений, содержащий квадраты
решения
    :return: Ничего
    """
    global operationCounter, DEBUG
    operationCounter += 1
    for x in range(xStart, xStart + width):
        for y in range(yStart, yStart + width):
            board[y][x] = color
    if DEBUG:
        print(f"Квадрат со стороной {width} ставят, координаты его
левого угла ({xStart}:{yStart})")
        printBoard(board)
    squares.append([xStart, yStart, width])


def sliceSquare(board, xStart, yStart, width, squares):
    """
    Функция отвечает за срез нижней стороны и правой в 1 клетку.
```

```
    Функция также уменьшает сторону последнего квадрата в частичном
решении на единицу.
    :param board: двумерный массив.
    :param xStart: Абсцисса левого верхнего угла квадрата для среза.
    :param yStart: Ордината левого верхнего угла квадрата для среза.
    :param width: Сторона квадрата для среза.
    :param squares: Массив частичных решений, содержащий квадраты
решения.
    :return: Ничего
    """
    global DEBUG
    for y in range(yStart, yStart + width):
        board[y][xStart + width - 1] = 0
    for x in range(xStart, xStart + width - 1):
        board[yStart + width - 1][x] = 0
    if DEBUG:
        print(f"Квадрат со стороной {width} уменьшают, координаты
его левого угла ({xStart}:{yStart})")
        printBoard(board)
    squares[-1][2] -= 1
    if squares[-1][2] == 0:
        squares.pop()


def backTracking(board, xStart, yStart, color, squares):
    """
    Функция бэктрэкинга. Находим максимальный размер квадрата
который можем вставить и вставляем его.
    Затем в цикле постепенно перебираем от найденого размера до
квадрата в одну клетку и ищем новую координату для вставки.
    Если такая есть, то для полученной координаты запускается эта же
функция,
    таким образом развивается это частичное решение. Если вставить
нельзя,
    то проверяем количество цветов в раскраске частичного решения
если оно меньше минимальной раскраски,
    то сохраняем решение и заменяем минимальную раскраску. В конце
цикла уменьшаем первый квадрат,
    который мы вставили.
    :param board: Двумерный массив.
    :param xStart: Абсцисса ещё не занятой области на столешницу.
    :param yStart: Ордината ещё не занятой области на столешницу.
    :param color: Текущий цвет
    :param squares: Массив частичных решений.
    :return: Ничего
    """
    global minColor, minBoard, minSquares
    # Поиск максимального размера квадрата для вставки
    maxSide = min(boardSize - yStart, boardSize - xStart)
    for x in range(xStart + 1, xStart + maxSide):
        if board[yStart][x]:
            maxSide = x - xStart
            break
    # Устанавливаем квадрат с найденным размером
    placeSquare(board, xStart, yStart, maxSide, color, squares)
    # Постепенно пребираем квадраты с меньшей стороной
    for n in range(maxSide, 0, -1):
```

```
        # Поиск новой координаты для вставки, ищем ещё не занятое
место
        flagFound = False
        for y in range(yStart, boardSize):
            if flagFound:
                y -= 1
                break
            for x in range(boardSize // 2, boardSize):
                if board[y][x] == 0:
                    flagFound = True
                    break
        # Если нашли снова запускаем бэктрэкинг, но уже с новой
координатой и увеличеным на единицу цветом
        if flagFound:
            if color + 1 != minColor:
                backTracking(board, x, y, color + 1, squares)
        else:
            # Найдено решение, если оно имеет раскраску меньше
текущей минимальной,
            # то сохраняем решение в глобальные переменные
            if color < minColor:
                minColor = color
                minBoard = deepcopy(board)
                minSquares = deepcopy(squares)
        # Уменьшаем вставленный квадрат, чтобы проверить остальные
варианты расстановки
        sliceSquare(board, xStart, yStart, n, squares)


if __name__ == "__main__":
    # Точка входа в программу.

    boardSize = int(input())  # Считывает входные данные.
    # Находит маштаб, если можно маштабировать задачу.
    factor = 1
    for i in range(2, boardSize + 1):
        if boardSize % i == 0:
            factor = boardSize // i
            boardSize = i
            break
    # Заводим первоначальный размер
    minColor = boardSize * boardSize + 1
    # Создаем двумерный массив столещницы в заданом маштабе
    board = [[0 for _ in range(boardSize)] for _ in
range(boardSize)]
    # Находим сторону максимального квадрата
    bigSquare = (boardSize + 1) // 2
    # Находим сторону двух смежных квадратов
    remainingSquare = boardSize // 2
    # Создаем массив квадратов частичного решения
    squares = []
    placeSquare(board, 0, 0, bigSquare, 1, squares)  # Устанавливаем
максимальный квадрат
    # Устанавливаем два квадрата смежных с максимальным
    placeSquare(board, bigSquare, 0, remainingSquare, 2, squares)
    placeSquare(board, 0, bigSquare, remainingSquare, 3, squares)
    # Запускаем рекурсивный бэктрэкинг
```

```
    backTracking(board, remainingSquare + (boardSize % 2),
remainingSquare, 4, squares)
    # Вывод дополнительной информации
    if DEBUG:
        print(f"Количество операций: {operationCounter}")
        print("Столещница с минимальным количество квадратов:")
        printBoard(minBoard)
        print("Основной ответ на поставленную задачу:")
    # Вывод результата работы программы
    print(minColor)
    for square in minSquares:
        print(square[0] * factor + 1, square[1] * factor + 1,
square[2] * factor)
```