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: 3и

Студент гр. 3388 _____ Трунов Б.Г.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2025

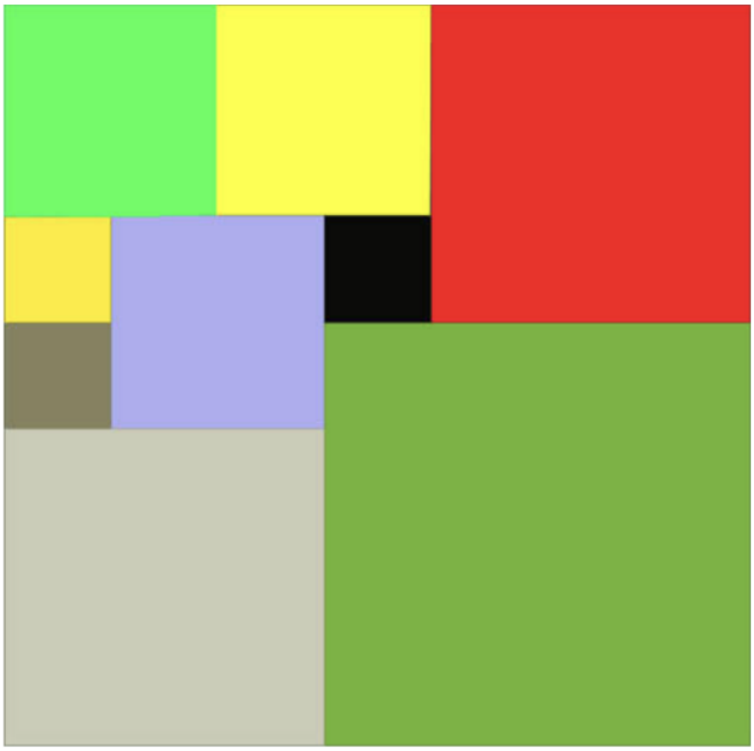
Цель работы:

Изучить теоретические основы алгоритма поиска с возвратом. Решить с его помощью задачу о разбиении квадрата. Провести исследование зависимости количества итераций от стороны квадрата.

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

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

Выполнение работы

Описание алгоритма:

Общее описание алгоритма:

Алгоритм нацелен на поиск минимального количества подквадратов, полностью покрывающих квадрат размером $N \times N$, сочетая метод backtracking с оптимизациями для сокращения вычислений. В зависимости от типа числа N выбирается начальная конфигурация: для простых значений используется стартовая схема из трёх крупных квадратов, а для составных применяется масштабирование решений, найденных для меньших размеров. Процесс перебора организован по принципу «от большего к меньшему» — сначала рассматриваются максимально возможные квадраты, что соответствует локальному оптимальному выбору.

На каждом шаге проверяется корректность размещения квадрата (отсутствие перекрытий с уже установленными). Ветви перебора, где число использованных квадратов превышает текущий рекорд, немедленно отсекаются. Для ускорения работы алгоритм задействует битовые маски для быстрой проверки условий, а также прерывает обработку заведомо неэффективных вариантов до их полного исследования. Эти оптимизации позволяют сохранить вычислительную эффективность даже для крупных значений N, минимизируя количество рассматриваемых комбинаций.

Основные этапы работы алгоритма:

- **Масштабирование исходного квадрата**
  - **Цель:** Уменьшение задачи для составных N (например, $N=k*m$).
  - **Действия:**
    - Находятся делители N.
    - Задача решается для квадрата с меньшей стороной $\min m \in \{m: m|N\}$ – m простое число, а коэффициент *upscaling'a* k не обязательно простое число.
    - Решение масштабируется обратно с коэффициентом k

- **Функции в коде**:
  - *scale_size(side_size : PositiveInt) -> tuple[PositiveInt]* – на вход подаётся сторона квадрата side*_size(N).* Функция ищет минимальный простой делитель числа *side_size* и возвращает кортеж из числа *m и числа upscaling coefficient.*

- **Инициализация начального разбиения**
  - **Жадная стратегия:**
    - В левый верхний угол помещается квадрат максимально возможного размера: Размер = $\lceil \frac{N}{2} \rceil$
    - Оставшиеся прямоугольники справа и снизу заполняются квадратами остаточного размера.
  - **Место в коде:**
    - В функции *solve* соответственно квадрат с размером *(N/2)* – квадрат с размером *start_pave_size*, квадраты остаточного размера – квадраты со стороной *remainder.*

- **Работа с битовой матрицей (*Bitboard)***
  - **Структура данных:**
    - Каждая строка матрицы кодируется битовой маской (1 – место занято, 0 – свободно)
  - **Функции в коде:**
    - *is_paved(self) -> bool – метод проверки замощения квадрата.*
    - *place_square(self, x_coord : PositiveInt, y_coord : PositiveInt, side_size : PositiveInt) -> None – метод постановки квадрата на BitBoard.*
    - *can_place_square(self, x_coord : PositiveInt, y_coord : PositiveInt, side_size : PositiveInt) -> bool – метод проверки возможности постановки квадрата на BitBoard.*

- **Перебор с отсечением *(Backtracking)***
  - **Шаги*:***
    - **Поиск первой свободной клетки*:***
      - Сканирование матрицы сверху вниз и слева направо.
    - **Перебор размеров квадратов:**
      - От максимально возможного до 1×1.
      - Для каждого размера проверяется возможность размещения.
      - При успешном размещении создаётся новая ветвь перебора.
    - **Отсечение ветвей:**
      - Если текущее количество квадратов превышает найденный минимум — ветка игнорируется.
  - **Функции в коде:**
    - *solve(side_size : PositiveInt, debug_mode : bool) -> SolveResult* функция поиска минимального замощения.
- **Сохранение оптимального решения**
  - **Лучшее решение:**
    - Сохраняется конфигурация с минимальным числом квадратов**.**
  - При обнаружении улучшения обновляется лучшее решение.

Оценка сложности алгоритма:

- **Временная сложность алгоритма**
  - Асимптотика $\mathbf{O(k^N)}$, где $1 < k < 2$.
    - Константа k зависит от структуры разбиений (Составные N имеют меньшее значение k, простые – большие).
  - **Доказательство:**
  - **Резкий рост сложности**
    - На каждом этапе алгоритм анализирует все допустимые размеры квадратов, которые можно разместить в текущей позиции (x,y). Количество вариантов определяется минимальным значением из $(N-x)$ и $(N-y)$, что в худшем случае приводит к $O(N)$ операций на шаг.
    - Без применения оптимизаций количество возможных комбинаций растёт катастрофически быстро — например, как функция $O(N^{N^2})$. Однако использование двух ключевых подходов позволяет смягчить эту проблему
  - **Оптимизации:**
    - Жадные стратегии — выбор локально оптимальных решений (приоритет крупных квадратов). И жадное начальное разбиение.
    - Масштабирование — сведение задачи к меньшему размеру для составных N. $k \approx 1.2\text{-}1.5$
    - Отсечение ветвей при достижении текущего минимума.
- **Пространственная сложность алгоритма**
  - Асимптотика $\mathbf{O(N^2 * k^N)}$, где k – та же константа что и для времени.

- **Доказательство:**
- Структуры данных:
  - *BitBoard*:
    - Хранит N целых чисел (битовые маски строк),
    - Занимает $O(N)$ памяти на состояние.
  - Список квадратов (squares):
    - В худшем случае (минимальные квадраты 1×1) содержит $O(N^2)$ элементов.
- Стек состояний:
  - Хранит пары (*BitBoard*, *squares*),
  - Максимальный размер стека: $O(k^N)$ (экспоненциальный рост).
- Дополнительные затраты по памяти:
  - Лучшее решение (*best_squares_com*): $O(N^2)$ памяти.

Визуализация

Для визуализации работы алгоритма была использована библиотека Pillow.

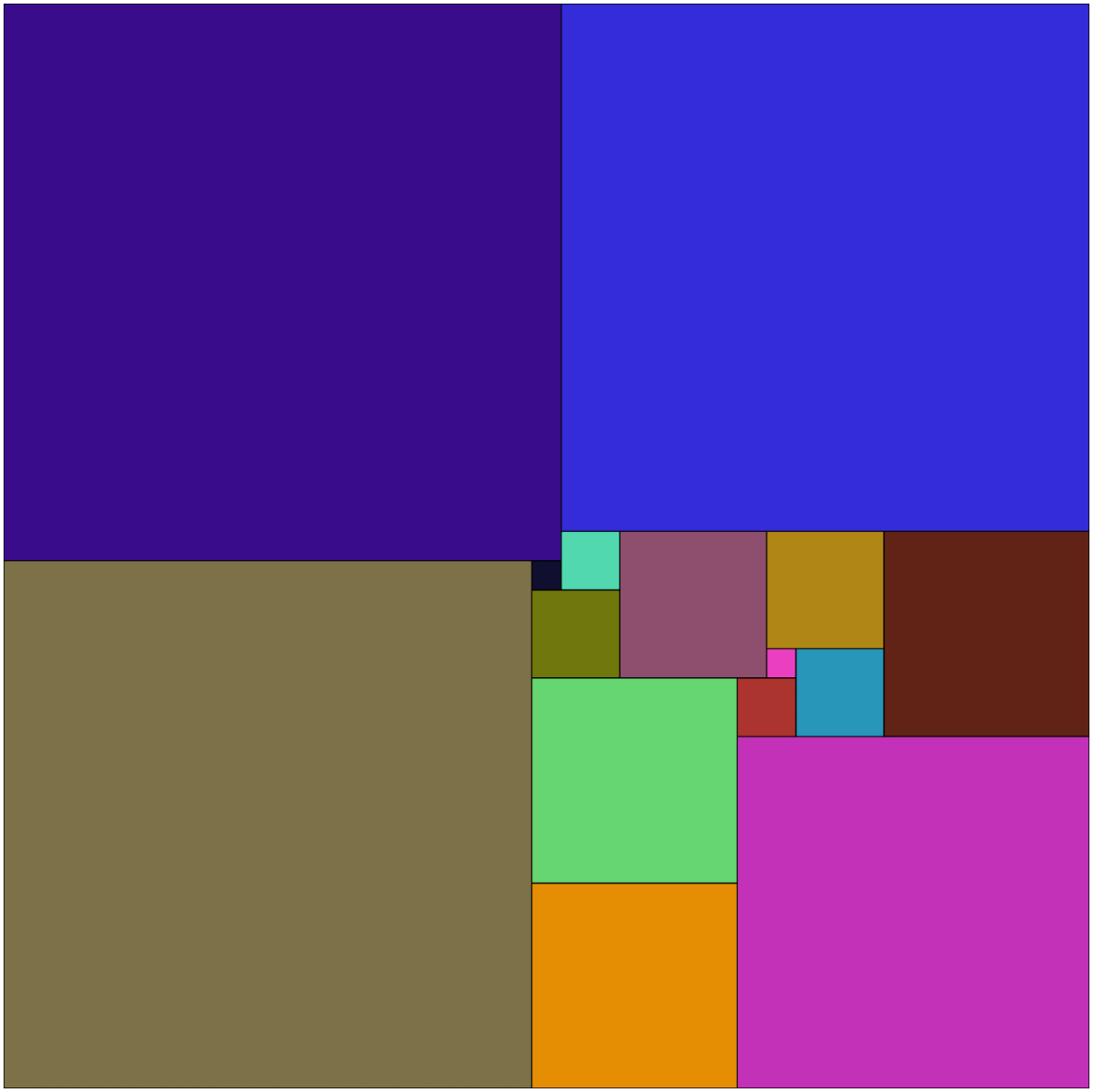


Рис. 1 Визуализация работы алгоритма.(N=37)

Тестирование

Таблица 1. Тестирование.

Входные данные	Выходные данные
7	9 1 1 4 1 5 3 5 1 3 4 5 2 4 7 1 5 4 1 5 7 1 6 4 2 6 6 2
25	8 1 1 15 1 16 10 16 1 10 11 16 5 11 21 5 16 11 5 16 16 10 21 11 5
26	4 1 1 13 1 14 13 14 1 13 14 14 13
31	15 1 1 16 1 17 15 17 1 15 16 17 1 16 18 1 16 19 4 16 23 3 16 26 6 17 16 3

	19 23 3 20 16 6 20 22 1 21 22 1 22 22 10 26 16 6
37	15 1 1 19 1 20 18 20 1 18 19 20 1 19 21 3 19 24 7 19 31 7 20 19 2 22 19 5 26 24 2 26 26 12 27 19 4 27 23 1 28 23 3 31 19 7

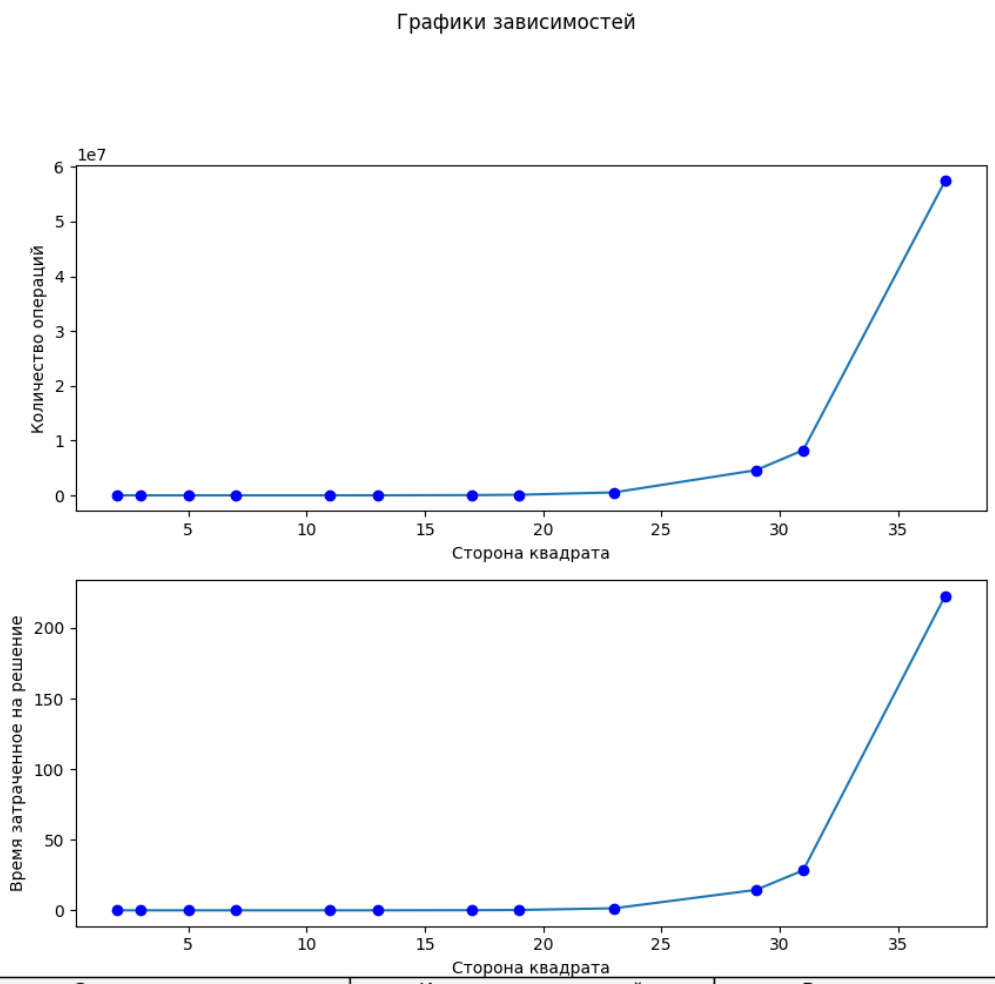
Исследование

В ходе лабораторной работы было проведено исследование зависимости количества итераций от стороны квадрата. В ходе исследования получились следующие результаты (рис. 1 и табл. 2).

Таблица 2. Зависимость количества итераций от стороны квадрата.

Сторона квадрата	Количество итераций
2	2
3	4
4	2
5	19

6	2
7	92
8	2
9	4
10	2
11	1776
12	2
13	5290
14	2
15	4
16	2
17	43801
18	2
19	103275
20	2
21	4
22	2
23	535267
24	2
25	19
26	2
27	4
28	2
29	4591530
30	2
31	8243190
32	2
33	4
34	2
35	19
36	2
37	57422881
38	2
39	4
40	2



Сторона квадрата	Количество итераций	Время замощения
2.0	2.0	2.7894973754882812e-05
3.0	4.0	1.8596649169921875e-05
5.0	19.0	5.030632019042969e-05
7.0	92.0	0.00018095970153808594
11.0	1776.0	0.003681659698486328
13.0	5290.0	0.011728525161743164
17.0	43801.0	0.10636138916015625
19.0	103275.0	0.25968360900878906
23.0	535267.0	1.4429750442504883
29.0	4591530.0	14.465620756149292
31.0	8243190.0	28.30267906188965
37.0	57422881.0	222.7324481010437

Рис. 2. Зависимость количества итераций и времени от стороны квадрата

Вывод

В ходе выполнения лабораторной работы был изучен и реализован алгоритм минимального замощения квадрата, основанный на комбинации методов масштабирования, жадных эвристик и итеративного поиска с возвратом.