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

Студентка гр. 3388 _____ Басик В.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2025

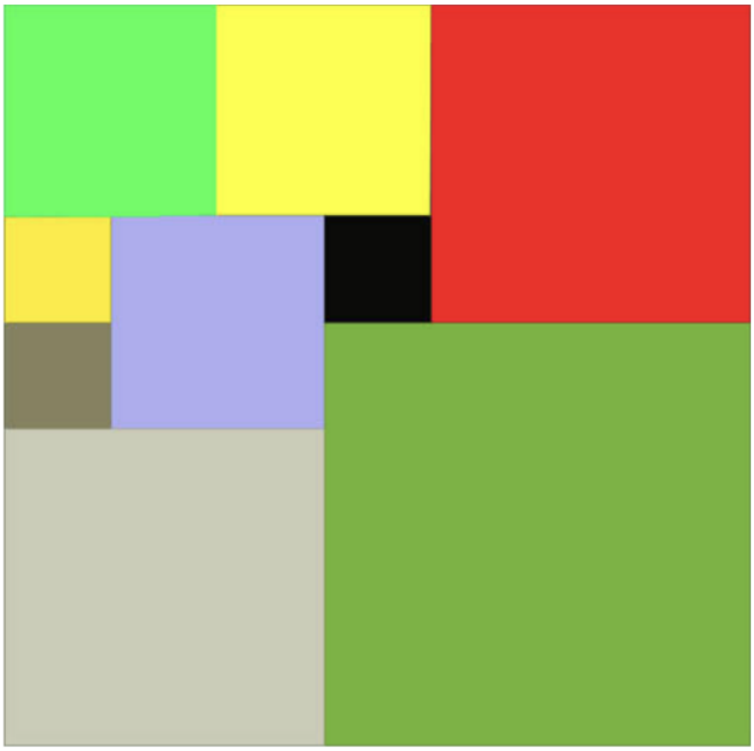
Цель работы:

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 ($1 \leq x,y \leq N$) 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

Выполнение работы

Описание алгоритма:

Общее описание алгоритма:

Алгоритм решает задачу покрытия квадрата размером N×N минимальным количеством подквадратов, используя метод итеративного перебора с отсечениями (backtracking). Начиная с начального разбиения (для простых NN — трёх крупных квадратов, для составных — масштабирования), он последовательно перебирает варианты размещения квадратов, от максимальных возможных размеров к минимальным. На каждом шаге проверяется возможность размещения квадрата без пересечений, а ветви с числом квадратов, превышающим текущий минимум, отсекаются. Это сочетание жадной эвристики, битовых оптимизаций и раннего прекращения неоптимальных ветвей позволяет эффективно находить решение, минимизируя вычислительные затраты.

Основные этапы работы алгоритма:

1. Масштабирование квадрата (Divide-and-Conquer)

- Цель: Уменьшить размер задачи для составных N.
- Механизм:
  - Если N имеет делители (напр., N = d × k), задача решается для меньшего размера d с последующим масштабированием результата в k раз.
  - Пример: Для N = 6 → d = 3, k = 2. Решение для 3×3 масштабируется в 6×6.
- Функции:
  - ScaleSize(N) – находит делители d и k.
  - upscale() – преобразует координаты и размеры квадратов из масштаба d в N.

2. Начальное разбиение для простых N

- Стратегия:
    1. Разместить главный квадрат размером (N+1)/2 в левом верхнем углу.
    2. Два квадрата размером N - (N+1)/2 размещаются в оставшихся углах.

3. Итеративный перебор с приоритетом больших квадратов

- Структура данных: Стек (stack<>) хранит состояния (BitGrid, список квадратов).
- Логика:
    1. Извлекается текущее состояние из стека.
    2. Находится первая свободная позиция через findFirstEmpty().
    3. Для позиции (x, y) перебираются квадраты от максимально возможного размера до 1×1.
    4. При успешном размещении (canPlace()) новое состояние помещается в стек.
- Ключевая оптимизация: Размещение больших квадратов сначала уменьшает глубину ветвления.

4. Отсечение неоптимальных ветвей (Pruning)

- Механизм:
    - Текущий минимум квадратов (minCount) обновляется при нахождении решения.
    - Ветви с current.size() ≥ minCount игнорируются.
- Эффект: Резко сокращает пространство поиска (до 90% для N > 10).

5. Битовые оптимизации (BitGrid)

- Структура:
    - Сетка представлена как vector<uint32_t>, где каждый бит соответствует клетке.

    - Проверка занятости: O(N) вместо O(N^2) за счет битовых масок.
- Операции:
    - canPlace(x, y, size): Проверка N битовых строк за O(N).
    - place() / unplace(): Модификация битовых масок.

6. Визуализация (saveImage)

- Реализация:
    - Генерация PNG-изображения через библиотеку libpng.
    - Каждый квадрат заливается случайным цветом, границы выделяются черным.
- Детали:
    - Масштабирование: 1 клетка = 50×50 пикселей.
    - Используется mt19937 для генерации цветов.

7. Бенчмаркинг

- Метрики:
    - Итерации: Счетчик в solveOriginal().
    - Время: Замер через <chrono> в main().

Описание функций и структур:

труктуры данных и функции

Структуры

1. Square – квадрат с координатами X,Y, и размера W
2. BitGrid
    - Инкапсулирует логику работы с битовой сеткой.
    - Методы:
        - canPlace(x, y, size) → bool
        - place(x, y, size) → void

- findFirstEmpty() → int (позиция в flat-представлении).

Ключевые функции

1. solveOriginal(N)
   - Возвращает: SolveResult (список квадратов + итерации).
   - Логика: Основной алгоритм с начальным разбиением и стековым перебором.
2. solveScaled(N)
   - Определяет необходимость масштабирования через ScaleSize().
   - Комбинирует решение для d×d с upscale().
3. ScaleSize(N)
   - Возвращает: pair(d, k), где d – делитель, k – коэффициент масштабирования.

Оценка сложности алгоритма:

Временная сложность алгоритма O(c^N) , где c > 1 — константа, зависящая от структуры разбиений. Это экспоненциальный рост, что подтверждается:

Доказательство:

1. Комбинаторный взрыв :
   - На каждом шаге алгоритм рассматривает все возможные размеры квадратов в текущей позиции
   - Для позиции (x,y) максимальный размер квадрата: min(N-x, N-y)
   - Число вариантов для каждого шага: O(N) (в худшем случае)
2. Глубина рекурсии :
   - Каждое размещение квадрата уменьшает площадь
   - Максимальная глубина: $O(N^2)$ (площадь квадрата)
3. Общая сложность :

- В худшем случае: O(N^{N²}) — но это верхняя оценка
- Практическая сложность: O(c^N) из-за оптимизаций ветвей и границ

Пространственная сложность алгоритма:
O(N² · c^N) (экспоненциальная), где c > 1 — константа, зависящая от структуры разбиений.

Детальный анализ:

- Стек stack:
  - Хранит пары (BitGrid, vector<Square>)
  - Каждый BitGrid занимает O(N) памяти (битовая маска для каждой строки)
  - Каждый vector<Square> в худшем случае содержит O(N²) элементов (для минимальных квадратов 1×1)
- Максимальный размер стека:
  - В худшем случае стек может содержать O(c^N) элементов (экспоненциальный рост состояний)
  - Каждый элемент стека занимает O(N + k) памяти, где k — текущее количество квадратов (до O(N²))
- Дополнительные структуры:
  - best (лучшее решение) занимает O(N²) памяти
  - current (текущее решение) в цикле занимает до O(N²) памяти

Визуализация

Для визуализации работы алгоритма была использована библиотека libpng.

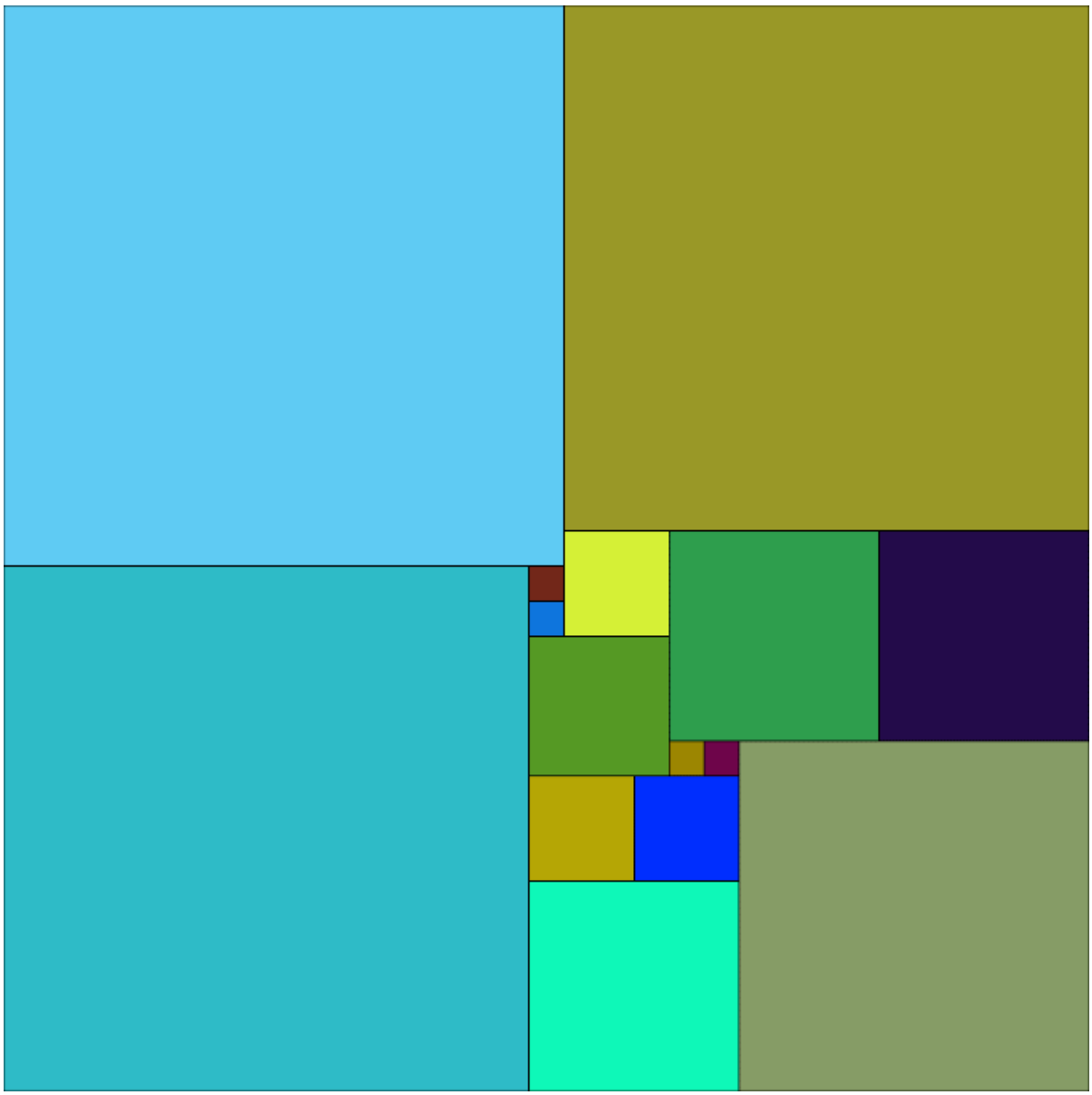


Рис. 1 Визуализация работы алгоритма.

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

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

Построим график зависимости количества итераций от стороны квадрата. Рассматривать будем только простые числа.

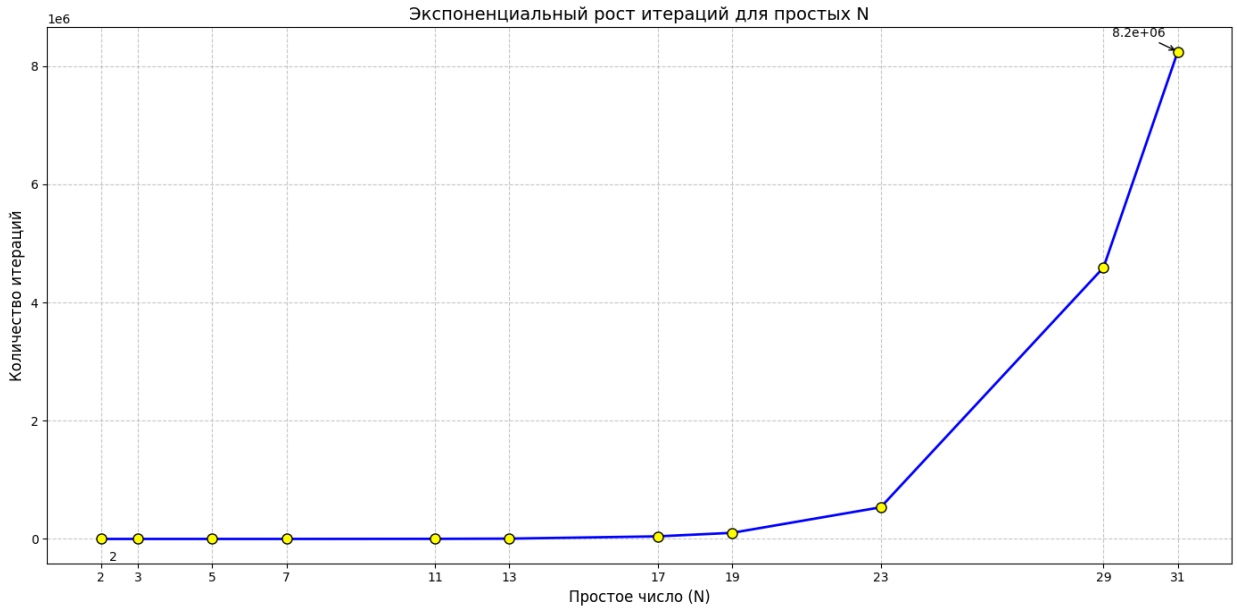


Рис. 2. Зависимость количества итераций от стороны квадрата

Вывод

В ходе лабораторной работы была написана программа с использованием итеративного метода backtracking. Также было проведено тестирование на различных входных данных по результатам, которого можно заключить, что число операций растет экспоненциально в зависимости от размера стороны квадрата.