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: 3р

Студент гр. 3388	____________________	Шубин П.А.
Преподаватель	____________________	Жангиров Т.Р.

Санкт-Петербург
2025

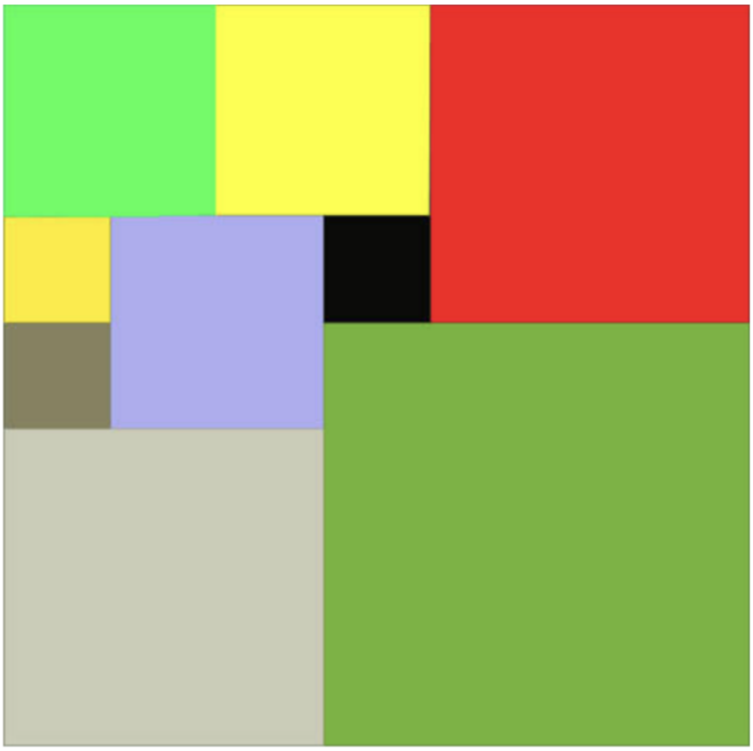
Цель работы:

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

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:

Рекурсивный алгоритм разбиения квадрата на минимальное количество подквадратов основан на методе backtracking (возврат к исходным данным). Основная идея алгоритма заключается в разбиении квадрата размером N×N на минимальное количество меньших квадратов, используя метод рекурсивного перебора с отсечениями (backtracking). Алгоритм стремится найти оптимальное покрытие квадрата, минимизируя количество используемых подквадратов.

Основные этапы работы алгоритма:

1. Масштабирование квадрата

Если размер квадрата N можно масштабировать (т.е. N имеет делители, отличные от 1 и самого себя), алгоритм уменьшает размер задачи, работая с меньшей сеткой.

Например, если N=6, его можно масштабировать до 3×3 с коэффициентом масштабирования 2. Это упрощает вычисления, так как задача решается для меньшей сетки, а результат затем масштабируется обратно.

2. Метод постановки трех начальных квадратов

Если масштабирование невозможно (например, N — простое число), алгоритм использует стратегию начального разбиения:

Размещает один большой квадрат размером (N+1)/2 в левом верхнем углу.

Размещает два меньших квадрата размером N/2 в оставшихся областях.

Это начальное разбиение помогает сократить пространство поиска и ускорить нахождение оптимального решения.

3. Рекурсивный перебор с отсечениями

Алгоритм рекурсивно перебирает все возможные варианты размещения квадратов, начиная с максимально возможного размера и уменьшая его до минимального.

Для каждой свободной клетки:

Определяется максимальный размер квадрата, который можно разместить в этой клетке без пересечения с уже занятыми областями.

Если квадрат успешно размещен, алгоритм продолжает поиск для оставшейся свободной области.

Если текущее количество квадратов превышает уже найденное оптимальное значение, алгоритм прекращает дальнейший перебор в этой ветке (отсечение).

4. Оптимизация через отсечения

Алгоритм отслеживает текущее количество квадратов и сравнивает его с лучшим найденным решением.

Если текущее решение уже хуже (использует больше квадратов), алгоритм прекращает дальнейший перебор в этой ветке, что значительно сокращает время выполнения.

5. Визуализация результата

После нахождения оптимального разбиения алгоритм визуализирует результат, создавая изображение, на котором каждый квадрат выделен своим цветом.

Это позволяет наглядно оценить, как квадрат был разбит на меньшие части.

6. Бенчмарк для анализа производительности

Алгоритм включает функцию бенчмарка, которая измеряет количество итераций и время выполнения для различных значений NN.

Это помогает оценить производительность алгоритма и его поведение на разных входных данных.

Описание функций и структур:

Основные структуры данных:

- Square — структура, описывающая квадрат:

Поля: x, y (координаты верхнего левого угла), size (длина стороны квадрата).

Функции:

- ScaleSize(gridSize int) (int, int): Определяет наибольший делитель gridSize для масштабирования сетки. Возвращает новый размер сетки и размер квадрата. Если масштабирование невозможно, возвращает исходный размер.
- placeInitialSquares(N int, occupied [][]bool) []Square: Размещает три начальных квадрата для оптимизации разбиения, если масштабирование невозможно.
- Solve(occupied [][]bool, current []Square, gridSize, scale int): Основная функция, реализующая рекурсивный алгоритм поиска разбиения. Обновляет список текущих квадратов и проверяет, является ли текущее решение оптимальным.
- findFirstFreePosition(occupied [][]bool, N int) int: Находит первую свободную клетку в сетке.
- canPlace(x, y, size int, occupied [][]bool) bool: Проверяет, можно ли разместить квадрат заданного размера в указанной позиции.
- placeSquare(x, y, size int, occupied [][]bool) Square: Размещает квадрат на сетке и обновляет занятую область.
- removeSquare(square Square, occupied [][]bool): Удаляет квадрат из сетки и освобождает занятую область.
- showGraphic(N int, squares []Square): Визуализирует результат разбиения квадрата на изображении.
- Benchmark(): Запускает бенчмарк для измерения производительности алгоритма на различных значениях N. Строит график зависимости количества итераций от размера сетки N.

Оценка сложности алгоритма:

Временная сложность

На каждом шаге:

1. Поиск первой свободной позиции: $O(N^2)$ (функция findFirstFreePosition).
2. Перебор возможных размеров квадрата (от maxSz до 1).
3. Проверка возможности размещения: $O(size^2)$ (функция canPlace).
4. Размещение/удаление квадрата: $O(size^2)$ (функции placeSquare, removeSquare).
5. Рекурсивный вызов для новой конфигурации.

Худший случай:

- Экспоненциальная сложность O(2^N) или O(k^N), где k — среднее количество вариантов на шаге.
- Причина: алгоритм перебирает все возможные комбинации размещения квадратов.

Оптимизации:

- Отсечение ветвей при len(current) >= minSquares.
- Начальное разбиение на 3 квадрата (сокращает поиск).
- Приоритет крупных квадратов (снижает количество шагов).

Пространственная сложность

Пространственная сложность определяется следующими компонентами:

- Сетка: Требуется $O(N^2)$ памяти для хранения булевого массива occupied, который отслеживает занятость клеток.
- Рекурсивный стек: Глубина рекурсии может достигать $O(N^2)$, так как алгоритм может попытаться разместить один квадрат в каждую клетку сетки.

- Текущие квадраты: Необходимо хранить список текущих квадратов, который может содержать до $O(N^2)$ элементов.
- Общая пространственная сложность: $O(N^2)$

Визуализация

Для визуализации работы алгоритма была использована библиотека gonum.org/v1/plot.

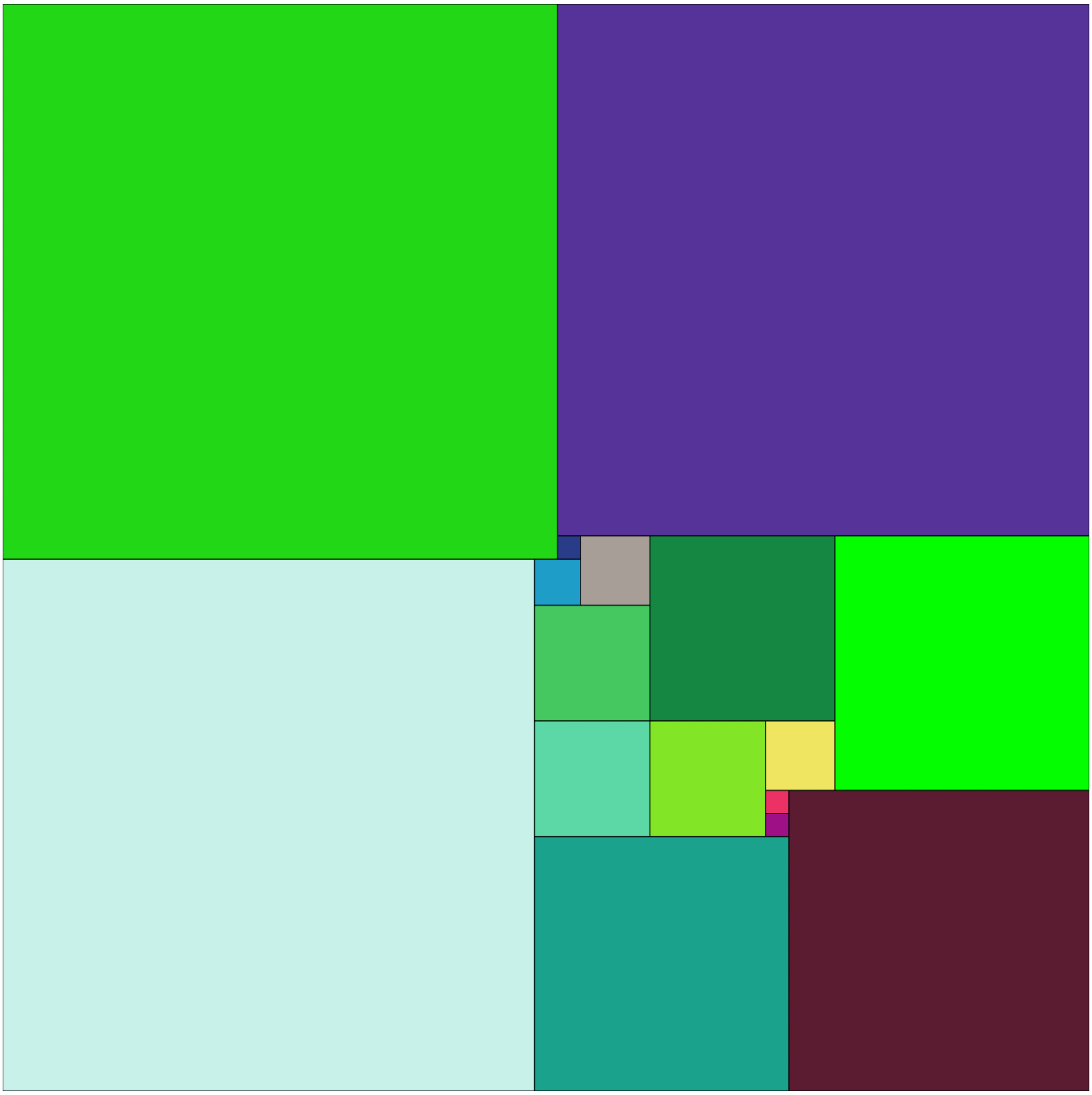


Рис. 1 Визуализация работы алгоритма.

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
15	6 1 1 10 1 11 5 6 11 5 11 1 5 11 6 5 11 11 5
20	4 1 1 10 1 11 10 11 1 10 11 11 10
37	15 1 1 19 1 20 18 20 1 18 19 20 2 19 22 5 19 27 11 20 19 1 21 19 3 24 19 8 30 27 3 30 30 8 32 19 6 32 25 1 32 26 1

	33 25 5

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	5
5	13
6	5
7	37
8	5
9	19
10	5
11	379
12	5
13	832
14	5
15	19
16	5
17	4626
18	5
19	12242
20	5
21	19
22	5
23	45087
24	5
25	277
26	5
27	19

28	5
29	306186
30	5
31	695883
32	5
33	19
34	5
35	3
36	5
37	3484074
38	5
39	19
40	5

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

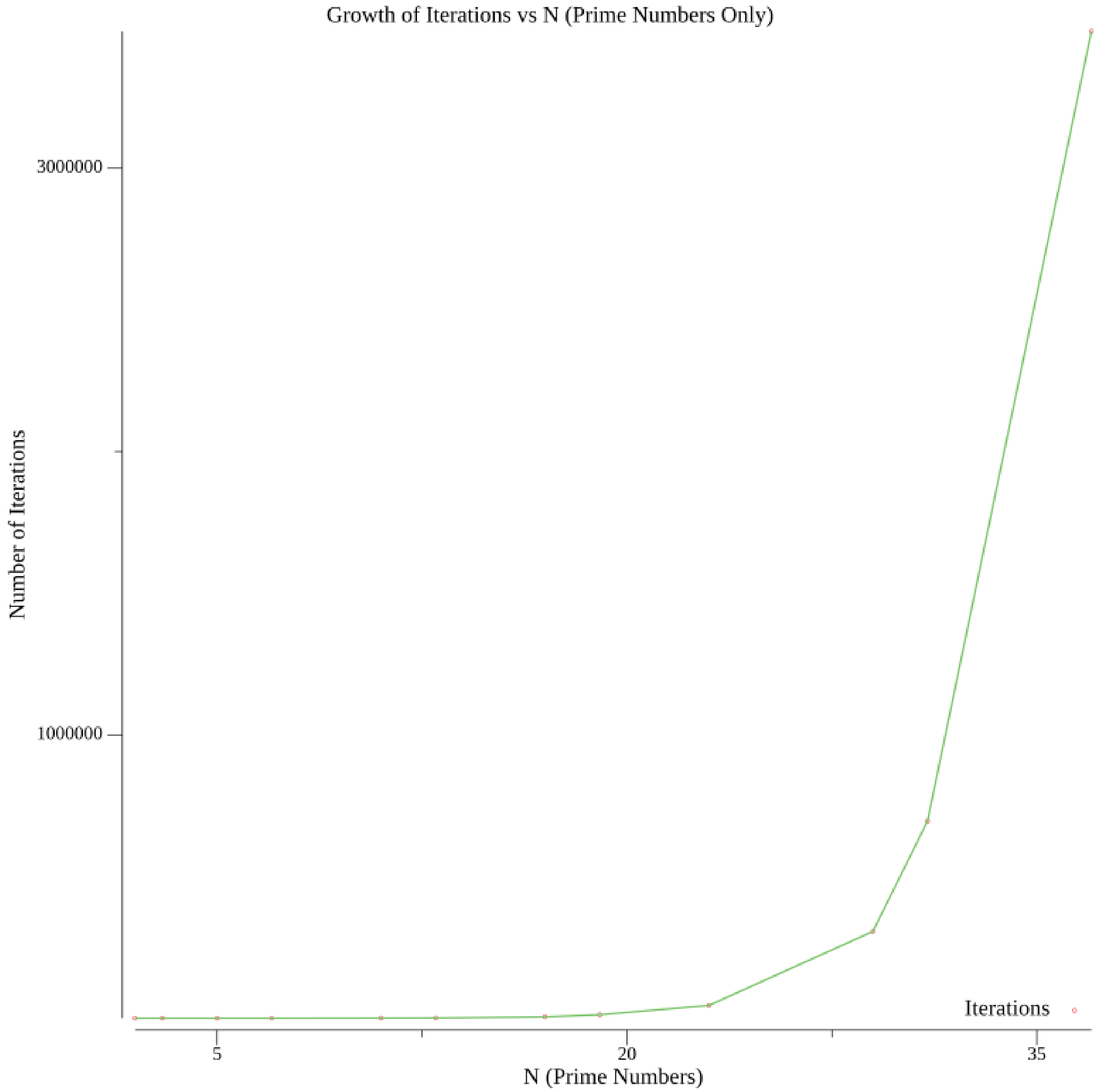


Рис. 2. Зависимость количества итераций от стороны квадрата

Вывод

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