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

ОТЧЕТ

по лабораторной работе №1 (Вар. 2р)

по дисциплине «Построение и анализ алгоритмов»

Тема: Поиск с возвратом

Студент гр. 3388	____________________	Лутфулин Д.А.
Преподаватель	____________________	Жангиров Т.Р.

Санкт-Петербург

2025

Цель работы

Разработка и анализ алгоритма разбиения квадрата размером $n \times n$ на минимальное количество непересекающихся меньших квадратов с помощью рекурсивного поиска с возвратом

Задание (Вариант 2р. Рекурсивный бэктрекинг. Исследование времени выполнения от размера квадрата)

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).

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

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

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,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

Для выполнения работы был использован алгоритм рекурсивного поиска с возвратом. Этот алгоритм на каждом вызове рекурсивной функции добавляет один квадрат и снова вызывает рекурсивную функцию, а если количество квадратов в решении уже превысило длину наилучшего решения, то происходит откат к более ранним решениям.

Описание рекурсивной функции:

Функция backtrack(cur_squares, empty, x, y) вызывается внутри главной функции quilts. Она принимает текущий список квадратов cur_squares, число empty – оставшаяся незаполненная площадь, и принимает координаты последней уже найденной заполненной точки x,y. Также функция пользуется двумерным массивом sq, размером стороны квадрата n и nonlocal массивом best_solution. Назначение – найти наименьшее разбиение квадрата на квадраты с меньшей стороной. Функция ничего не возвращает, но изменяет массив best_solution.

Оптимизации алгоритма:

Для оптимизации алгоритма были использованы следующие методы:

1. В качестве первых трёх квадратов предполагаются 3 квадрата:
    - Квадрат стороны (n + 1) // 2 в точке [0,0]
    - Квадрат стороны n//2 в точке [(n + 1) // 2, 0]
    - Квадрат стороны n//2 в точке [0, (n + 1) // 2]

Например для квадрата 7*7 получим такое разбиение:

```
[4, 4, 4, 4, 3, 3, 3]
[4, 4, 4, 4, 3, 3, 3]
[4, 4, 4, 4, 3, 3, 3]
[4, 4, 4, 4, 0, 0, 0]
[3, 3, 3, 0, 0, 0, 0]
[3, 3, 3, 0, 0, 0, 0]
[3, 3, 3, 0, 0, 0, 0]
```

2. Если сторона квадрата – не простое число, то находим решение для квадрата со стороной наименьшего делителя этого числа, а потом масштабируем его до исходного n. Например, разбиение квадрата 14*14 – это разбиение квадрата 2*2, увеличенное в 7 раз.
3. Основная оптимизация перебора – бэктрекинг. Если найденный массив cur_squares длиннее, чем массив best_solution, или мы перебрали все возможные варианты, то алгоритм возвращается к предыдущему шагу, удаляя последний поставленный квадрат.

4. Введена переменная empty – количество оставшихся пустых клеток. Она применяется, если мы пытаемся поставить квадрат с площадью больше, чем её значение – в таком случае попытка сразу отбрасывается, без перебора клеток. Также, если значение empty = 0, то можно не пытаться поставить ещё один квадрат, а сразу выходить из функции, так как квадрат уже заполнен.

Способ хранения частичных решений:

Частичные решения хранятся в двумерном массиве sq размера n*n, массиве текущего решения cur_squares и массиве best_solution. Массив best_solution был сделан nonlocal для того, чтобы можно было копировать в него массив cur_squares, когда тот становится оптимальным решением. Для sq и cur_squares это не требуется, так как с ними не выполняется операция присваивания.

Оценка сложности по времени и памяти:

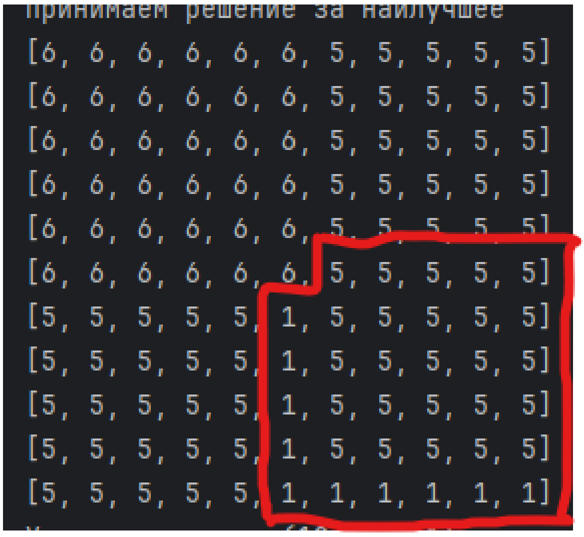
Нахождение простого делителя – $O(n^{1/2})$

Здесь и далее под n будет подразумеваться наименьший простой делитель числа, так как решение для составного числа является масштабированным решением для простого числа.

Рисование квадрата – $O(n^2)$

Построение 3-х начальных квадратов – $O(n^2)$.

Заметим, что первое найденное решение будет с большим квадратом со стороной (n//2) и остальными квадратами 1*1. Это ровно n решений. Пример для квадрата 11*11 ниже:



Таким образом наилучшее решение уже размером n, а значит будет не более чем n заходов в рекурсию – остальные будут обрубаться сразу же, так как есть более оптимальное решение.

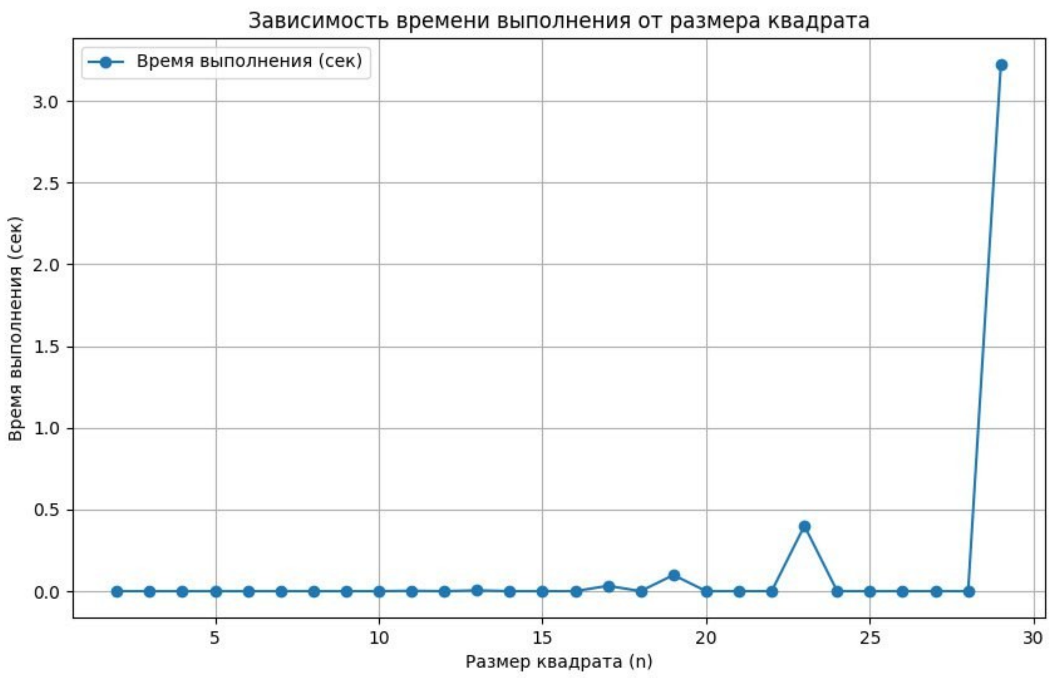
На каждом шаге мы сначала пытаемся найти следующую пустую точку. Так как квадрат заполняется слева направо и сверху вниз, будет проверяться примерно одна строка – в среднем случае O(n). Далее мы ставим квадрат размером не более чем $(n-1)^2$, рекурсивно вызываем функцию и убираем квадрат. Таким образом, 1 заход в функцию занимает $O(n) + O(n^2) + O(n^2) \sim O(n^2)$.

Итого имеем рекурсию глубины n, каждый шаг которой сложностью в O(n^2). Значит, сложность алгоритма O).

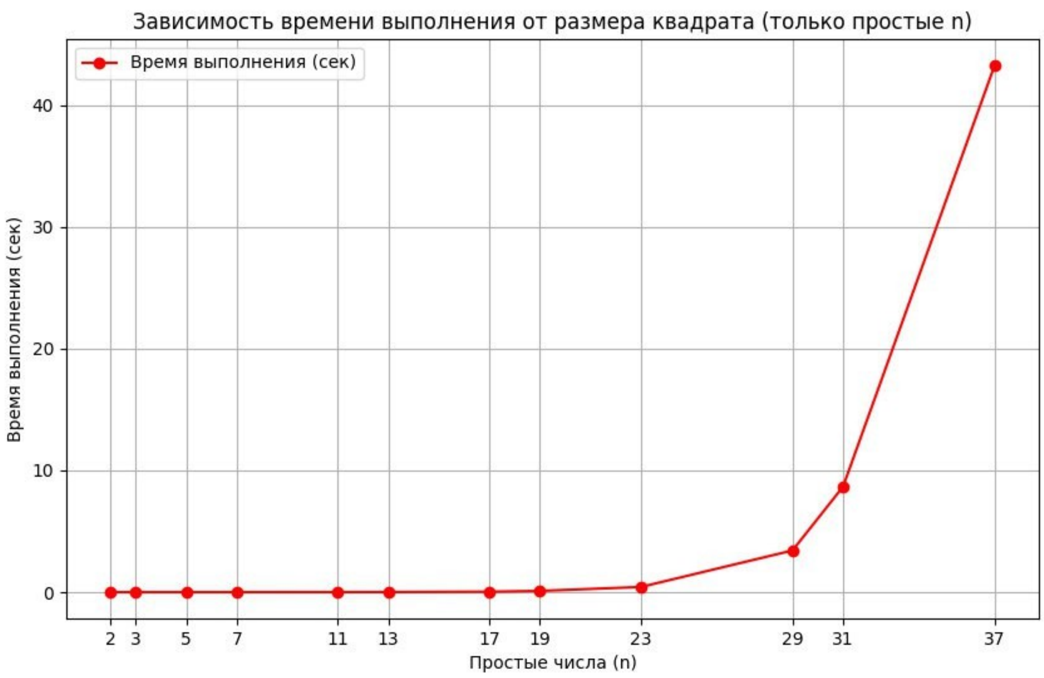
Оценим сложность по памяти. Массивы sq best_solution и cur_squares ни на каком из этапов не копируются, их сложность $O(n^2)$, O(n) иO(n) соответственно. Так как глубина рекурсии - n, сложность для остальных переменных O(n). Итого получаем сложность $O(n^2)$.

Исследование времени выполнения :

Построим график зависимости времени выполнения от n



Так как решение для составных чисел сводится к масштабированию простого числа, решения с составными числами не составляют большого интереса. Поэтому имеет смысл рассмотреть график, на который нанесены только простые числа



Вывод

Разработан и проанализирован алгоритм разбиения квадрата n×n на минимальное количество непересекающихся квадратовс использованием поиска с возвратом. Оценена сложность алгоритма по времени – O) и по памяти - $O\left(n^2\right)$. Исследовано время работы алгоритма