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

ОТЧЕТ

по лабораторной работе №1 (Вар. 1и)
по дисциплине «Построение и анализ алгоритмов»
Тема: Поиск с возвратом

Студент гр. 3388	____________________	Сабалиров М.З.
Преподаватель	____________________	Жангиров Т.Р.

Санкт-Петербург
2025

Задание (Вариант 1и. Итеративный бэктрекинг. Выполнение на Stepik двух заданий в разделе 2)

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 итеративного поиска с возратом. Это переборный алгоритм применяемый при наличии частичных решение и критериев их сравнения. Т.е. на каждом шаге перебора (в данном случае добавление нового квадрата) происходит сверка с критериями перебора. Если это решение полное — сохраняем его. Если оно частичное — перебираем дальше его потомков. Если оно не подходит под критерии, то его потомки больше не буду перебираться.

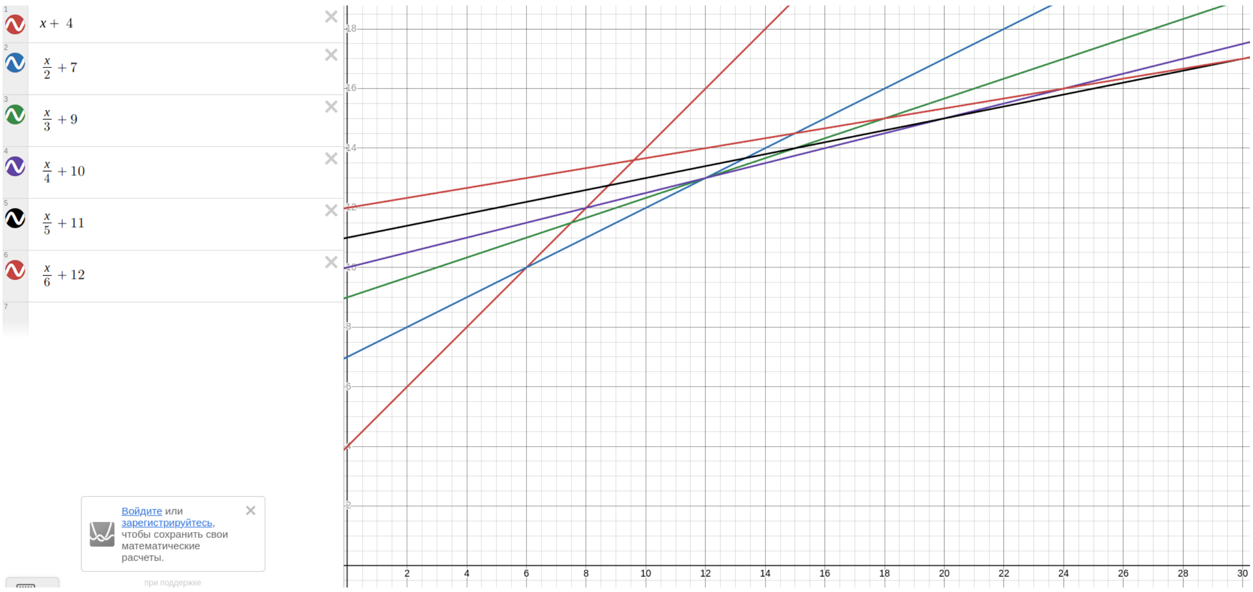
Написанное решение работает для натуральных числе от 2 до 40 **(было пройдено 3 модуля вместо 2-х)**. Для оптимизации перебора используются некие математические соображение. Факты без формальных доказательств. Предположения о простых числах тестировались на заданном отрезке:

1. Составные числа имеют оптимальное решение, параметры которого вычислимы: (из рассмотрения исключены четные числа [их всегда можно поделить на 4 квадрата]). Максимальная сторона внутреннего квадрата будет равно $M = (N / p)*(p / 2 + 1)$, где p — минимальный делитель N. «Нижняя» (не минимальная) m = Максимальная - N / p. А минимальное количество квадратов $min = (M/m)*2 + (m/(M-m))*2 + 2$.

2. Для простых чисел $M = N / 2 + 1$; $m = N — M$

3. Верхняя оценка минимального количества квадратов для простого числа = MIN(N + 4, N / 2 + 7, N / 3 + 9, N / 4 + 10, N / 5 + 11, N / 6 + 12)

Для примера оценка N + 4 является верной, так как мы можем взять 1 квадрат N / 2 + 1 и 3 N - (N / 2 — 1). Тогда при правильной расстановке (которая всегда возможна) Останутся две полоски высотой 1 и длиной N / 2. Остальные оценки, аналогично. Для разных N разные оценки могут быть оптимальными.



Работа алгоритма:

1. Вычисление оптимальных сторон и верхней оценки числа квадратов. Оценка уменьшается на константу, что бы не пропустить более короткое решение (для простых чисел) .

2. Расстановка оптимальных квадратов

3. Сужение задачи до заполнения квадрата М с начальным условием (в виде выпирающей части главного квадрата, если есть таковая)

4. Инициализация стека, комбинаций квадратов

5. Поиск точки с минимальными координатами для установки квадрата. Запись в стек всех возможных квадратов на этой точке. После этого алгоритм рассматривает следующую расстановку в стеке (другие точки не берутся, так как нет разницы, все равно все возможные [подходящие критериям] комбинации будут рассмотрены).

6. Если найдется хоть одно решение, то программа выводит ответ и завершается. Иначе верхний предел размера комбинации увеличивается на 1. И все повторяется с шага 2.

Способ хранения частичных решений:

Частичное решение:

```
typedef struct {
    size_t curr_square;
    size_t matrix[MAX_N];
    square_t squares[MAX_ASSESMENT];
    size_t len;
    size_t next_x;
    size_t next_y;
} combination_t;
```

Хранит текущую площадь, псевдо-матрицу (вместо строк числа, которые при работе читаются как двоичные), список квадратов в данном решении и их количество, точку с которой следует начать искать свободное место дочерним комбинациям

Оптимизации алгоритма:

1. Подсчет оптимального размера сторон и верхнего предела длинны решения исходя из N

2. На каждом шаге выбирается только одна точке, а не все доступные (иначе получим одинаковые решения с разным порядом)

3. В приоритете проверяются решения с большой площадью (помещаются в стек последними)

4. Ограничение по длине решения

5. При нахождении решения программа завершает работу

6. Сужение задачи до квадрата с меньшей стороной, но с начальным условием (не всегда начальное условие будет присутствовать)

7. Итерация по плоскости не с начала с точки + 1 родительского частичного решения

Оценка сложности и памяти:

Пусть минимум отличается от оценки на константу, она учитываться не будет. Так же не смотря на то, что задача сокращена до более меньшего квадрата для оценки будет использовано N.

На каждом шаге находится одна точка (Не более N^2 операций [А в подавляющем большинстве случаев сильно меньше]). Проверка возможности квадрата за O(1). Так как квадрат заполняется сверху вниз, а строка проверяется побитовыми операциями. На точку не более N — 1 квадратов. Длина решения линейно зависит от N. Тогда N позиций, поиск места N^2 и вариантов N — 1. => Сложность O(N^(N^2)). Не N^3, т. к. для точку для N -1 квадратов ищем только 1 раз. В действительности получается еще меньше, потому что поиск новой точки у дочерней комбинации начинается с точки родительской комбинации + 1.

Оценка памяти:

Размер стека на каждом шагу увеличивается не более чем на N — 1, на одно частичное решение выделяется O(N) пямяти. Тогда для N позиций потребуется O(N^3) памяти.