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 _____ Беннер В.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2025

Цель работы:

Изучить метод поиска с возвратом и разработать алгоритм, который эффективно применяет этот метод для решения задачи разбиения квадрата на меньшие квадраты.

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 решения задачи применялся алгоритм итеративного поиска с возвратом (backtracking). Этот переборный метод позволяет находить решения, постепенно расширяя частичные результаты. На каждом шаге (в данном случае при добавлении нового квадрата) проверяется соответствие текущего решения заданным критериям. Полные решения сохраняются, частичные расширяются, а несоответствующие отбрасываются, чтобы избежать ненужного перебора.

Написанное решение работает для натуральных числе от 2 до 40 **(пройдено 3 модуля вместо 2-х)**.

Работа алгоритма:

1. Определение ориентировочного решения: Сначала алгоритм определяет, сколько примерно квадратов потребуется для решения задачи. Также вычисляются размеры самых крупных квадратов, с которых начнётся заполнение.

2. Начало заполнения: Квадрат заполняется несколькими большими квадратами. Это создаёт основу, с которой дальше будет работать алгоритм.

3. Поиск свободных мест и добавление новых квадратов: Алгоритм ищет пустые места на основном квадрате. Для каждого такого места подбираются все возможные квадраты, которые могут туда поместиться. Эти квадраты добавляются в список рассматриваемых вариантов (в "стек").

4. Проверка каждого варианта (комбинации квадратов): Из списка вариантов берётся текущая комбинация квадратов и проверяется, заполнила ли она весь квадрат.

- Если да - решение найдено! Выводится информация о том, какие квадраты использовались.
- Если нет - проверяется, не слишком ли много квадратов уже использовано в этом варианте. Если квадратов слишком много, то дальнейшее

рассмотрение этого варианта прекращается - он не приведёт к оптимальному решению.

5. Перебор вариантов: Алгоритм перебирает варианты, пока не найдёт решение, или пока не убедится, что с текущими ограничениями решения нет. Если перебраны все варианты и решение не найдено, то алгоритм немного ослабляет ограничения (разрешает использовать больше квадратов) и начинает перебор заново.

6. Повторение до нахождения решения: Шаги 3-5 повторяются до тех пор, пока не будет найдено разбиение основного квадрата на квадраты меньшего размера.

Способ хранения частичных решений:

Класс SquareCombination (этот класс представляет частичное решение задачи разбиения квадрата)

Он хранит:

- area: Общая площадь заполненных квадратов.
- matrix: Состояние заполненности квадрата (используется для проверки пересечений).
- squares: Список квадратов, входящих в текущую комбинацию.
- next: Точка, с которой следует начинать поиск места для следующего квадрата.

Оптимизации алгоритма:

1. Определение цели и стратегии: Сначала вычисляются оценки для размеров квадратов, которые будут использоваться, и определяется максимальное количество квадратов, которые нужно использовать для решения.

2. Устранение дубликатов: На каждом шаге выбирается только одна точка для добавления нового квадрата, чтобы избежать генерации одинаковых решений, отличающихся только порядком квадратов.

3. Приоритезация больших квадратов: Алгоритм стремится в первую очередь размещать квадраты большего размера, поскольку комбинации с меньшим количеством больших квадратов считаются более перспективными. Это реализуется добавлением вариантов с большими квадратами в стек последними, чтобы они извлекались первыми.

4. Ограничение глубины поиска: Существует ограничение на количество квадратов в одном варианте решения. Если вариант превышает это ограничение, он отбрасывается, чтобы не тратить время на бесперспективные ветви поиска.

5. Ранний выход: Как только найдено решение, алгоритм завершает свою работу, даже если потенциально существуют другие решения.

6. Сокращение области поиска: Задача упрощается путем рассмотрения только части исходного квадрата, а не всего целиком. При этом учитывается начальное условие - уже заполненная часть (если она есть).

7. Эффективный перебор: При поиске свободного места для нового квадрата перебор начинается не с самого начала, а с точки, следующей за тем местом, где был расположен последний квадрат в частичном решении. Это позволяет избежать повторного просмотра уже проверенных областей.

Оценка сложности и памяти:

1. Временная сложность: Максимальная длинна комбинации линейно зависит от N, количество вариантов зависит от N^2

=> сложность будет O(N^2)

2. Оценка памяти: размер стека всегда не больше чем N*(N-1), размер комбинации не больше, чем N

=> общая оценка памяти будет N^3

Вывод:

Разработан алгоритм, основанный на поиске с возвратом, для разбиения квадрата на меньшие квадраты. Алгоритм использует оценки

размеров квадратов и ограничения на их количество, чтобы сузить область поиска и быстро находить подходящие решения. Использование приоритета для больших квадратов и отсечение бесперспективных вариантов помогло ускорить работу алгоритма.