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

Студентка гр. 1304 _____ Хорошкова А.С.

Преподаватель _____ Шевелева А.М.

Санкт-Петербург

2023

Цель работы.

Изучение алгоритма поиска с возвратом (бэктрекинга). Решение задачи разбиения квадрата на минимальное количество квадратов меньшего размера.

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 как показано на Рисунок 1.

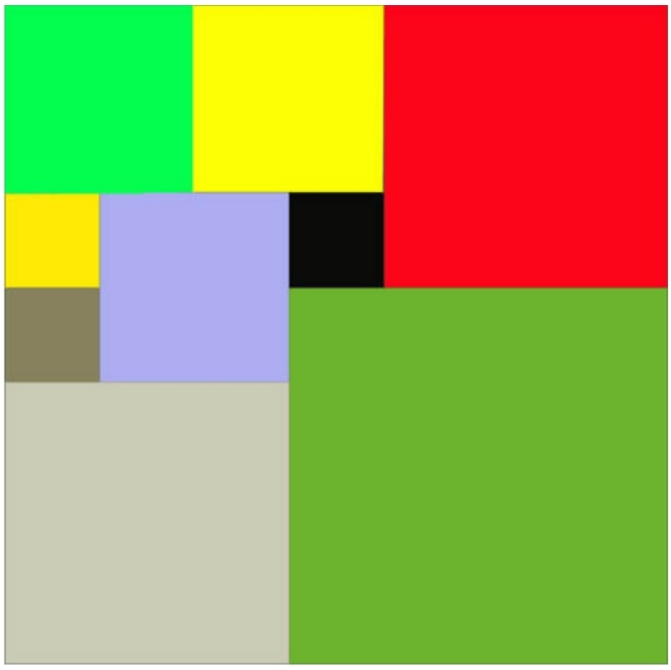

Рисунок 1 — построение столешницы 7x7

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 ($1 \le x,y \le N$) 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.

Поиск оптимального разбиения квадрата решается с помощью рекурсивного алгоритма бэктрекинга:

Определяются границы размеров квадрата, который будет начинаться на этом шаге в этой клетке от 1 до maxSize. Первый размер будет maxSize.

1. Вставляется квадрат текущего размера, если это возможно.
2. В следующей клетке начинается новый алгоритм вставки (начиная с пункта 1)
3. Удаляется квадрат, вставленный в пункте 2, если он был создан.
4. Если достигнут минимальный размер квадрата, то шаг заканчивается. Если «следующих» клеток нет, то шаг заканчивается и , если количество задействованных квадратов меньше найденного ранее, матрица с расположениями квадратов сохраняется, а в переменную минимального количества квадратов записывается новое значение.
5. Начинаем пункт 2. с размером на единицу меньше предыдущего.

В этот алгоритм были добавлены следующие изменения для оптимизации:

1. находится минимальный простой делитель p числа *N,* все дальнейшие действия происходят с матрицей p*p, а на последнем этапе результат выдаётся с учётом шага N/p.
2. Если p=2, то квадрат делится на 4 части и алгоритм заканчивается.
3. В матрицу p*p добавляются 3 квадрата так, как показано на Рисунок 2.

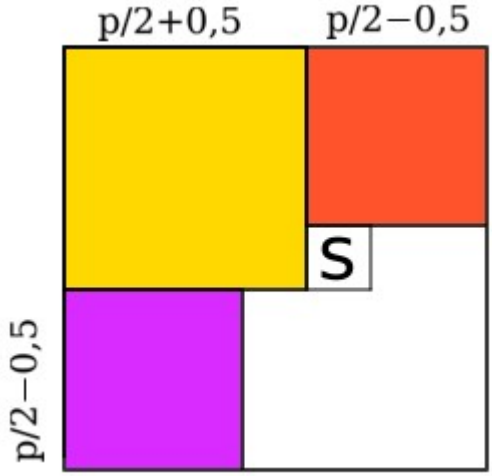

Рисунок 2 — начальное разрезание квадрата

4. Алгоритм бэктрекинга начанается с квадрата s, показаном на Рисунок 2.
5. Если во время алгоритма бэктрекинга шаг количество квадратов сравнялось с ранее найденным минимальным количеством квадратов, то этап алгоритма заканчивается.
6. MaxSize вычисляется как минимум из сторон возможной для заполнения текущим квадратом области.

Описание функций и структур данных.

Класс хранения и обработки назван SquarePuzzle. Класс имеет поля:

p — размер матрицы (минимальный простой делитель числа N).

step — *шаг* N/p.

square[][] — матрица для заполнения. В матрице квадраты хранятся в формате: 0 — клетка пуста, -1 — клетка занята, но начало квадрата не в этой клетке, 1≤*N* — клетка занята, начало квадрата в этой клетке, размер квадрата равен *N.*

count — счётчик, отображающий текущее количество квадратов в square[][].

squareTmp[][] — буферная матрица для заполнения, нужна для алгоритма бэктрекинга. Хранит оптимальное разрезание.

minCount — минимальное количество квадратов для разрезания.

Класс имеет методы:

SquarePuzzle() — конструктор, находит минимальный делитель p и шаг step, выделяет память для матриц square[][] и squareTmp[][] и запускает алгоритм поиска оптимального разрезания puzzle().

puzzle() — Запуск разрезания квадрата, добавление начальных квадратов (Рисунок 1), отработка случая p=2, запуск бэктрекинга при необходимости с нужного места.

BacktrackHelper() — поиск следующего квадрата для запуска бэктрекинга, сохранение результата при необходимости. Является вспомогательным для метода backtracking().

backtracking() — алгоритм бэктрекинга, описан в разделе выше.

addSquare() — добавление нового квадрата в матрицу square[][] Обновление счётчика count.

addSquareWithChecking() — добавление нового квадрата в матрицу square[][] с проверкой возможности этого добавления. Обновление счётчика count. Возвращает true — добавлено, false — невозможно добавить. Является вспомогательным для backtracking(). Для добавления квадрата вызывает addSquare().

deleteSquare — удаление квадрата из матрицы square[][]. Обновление счётчика count.

cloneTmp() — клонирование текущего варианта разрезания в буферную матрицу

printResult() — печать результата в требуемом формате.

Также создан публичный метод Main, содержащий единственный метод main() – точки входа в программу. В методе main() считывается размер квадрата N, запускается алгоритм нахождения оптимального разрезания с помощью класса SquarePuzzle и печатает результат с помощью метода printResult().

Выводы.

В ходе работы был рассмотрен рекурсивный алгоритм бэктрекинга. Была написана программа, использующая алгоритм бэктрекинга для нахождения варианта разбиения квадрата NxN на квадраты меньшего размера таким образом, чтобы количество квадратов было наименьшим из возможных. Также были добавлены изменения в алгоритм бэктрекинга для оптимизации решения задачи. Изменения позволили сводить составные N к простым числам p, $p \leq N$, алгоритм стал требовать меньше повторений, так как работал на площади меньшего размера. Также были вынесены в отдельный случай квадраты с чётной стороной N, а для остальных квадратов были заданы начальных квадраты, общие для всех нечётных случаев, что также помогло уменьшить площадь работы алгоритма. Для уменьшения вариантов перебора были введены максимальные стороны квадрата и добавлена преждевременная остановка для заведомо неэффективных вариантов разрезания. В результате алгоритм успешно выполняется для $2 \leq N \leq 40$ за отведённое время. Таким образом, была выполнена лабораторная работа №1 и два её «дополнения».