**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**

|  |  |  |
| --- | --- | --- |
| Студент гр. 1304 |  | Макки К.Ю. |
| Преподаватель |  | Шевелева А.М. |

Санкт-Петербург

2023

**Цель работы**

Изучить бэктрекинг и применить его для решения задачи разбиения квадрата размером n на минимальное количество квадратов меньшего размера с максимальным размером n - 1.

**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.

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 ≤ N ≤ 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≤ *x,y* ≤ *N*) и длину стороны соответствующего обрезка(квадрата).

**Выполнение работы.**

В начале программы пользователь должен ввести размер квадрата который хранится в переменной n. После вызывается конструктор класса SISS (Simple Imperfect Squared Squares) который принимает размер квадрата n. Этот конструктор имеет поля:

* square\_size, в котором хранится размер матрицы квадрата нужный для оптимальной работы программы, а не введенный пользователем;
* proportion переменная нужна для правильного вывода ответа;
* cur\_count и optimal\_count переменные нужные для хранения количества квадратов в течение рекурсии и их оптимального количества;
* 2 двухмерных вектора для хранения квадрата в виде int.

Перовое что выполняется в конструкторе это Min\_Prime\_Nums( ) это функция, которая находит меньшее простое число для размера квадрата и записывает его в качестве square\_size в целях уменьшения массива, а также меняет proportion чтобы при выводе вернуть изначальный размер введенный пользователем. Затем создаются два вектора с размером square\_size(минимальный размер например в месте 40 будет 2). Размер первого и самого большого квадрата считается по формуле (square\_size + 1) / 2 потом заполняется матрица первыми 3 квадратами самый большой в верхнем левом углу и 2 квадрата square\_size — (square\_size+1) / 2. Затем запускается backtracking( ) который ищет оптимальный размер квадрата, помещающегося в оставшейся области и ставит его в левый верхний угол. Если optimal\_cunt <= cur\_count завершается backtracking. В случае если предыдущее условие не проходит, в функции устанавливается размер квадрата с maxSize до 1. Если клетка пустая вставляется квадрат размера maxSize, после в следующей клетке начинается новый алгоритм вставки и удаляется клетка которую мы только вставили. В случае если мы дошли до минимального размер квадрата шаг заканчивается. Если клеток с меньшем размером нет, то шаг заканчивается и если cur\_count меньше optimal\_count, матрица с расположениями квадратов сохраняется, а в переменную минимального количества квадратов записывается новое значение. Потом в следующей клетке начинается новый алгоритм размером на единицу меньше предыдущего. После нахождения лучшего ответа печатается результат в терминал.

**Описание функций и структур данных.**

Класса хранениие SISS (Simple Imperfect Squared Squares)

Класс имеет поля:

* square\_size, в котором хранится размер матрицы квадрата нужный для оптимальной работы программы, а не введенный пользователем;
* proportion переменная нужна для правильного вывода ответа;
* cur\_count и optimal\_count переменные нужные для хранения количества квадратов в течение рекурсии и их оптимального количества;
* 2 двухмерных вектора для хранения квадрата в виде int.

Класс имеет методы:

* SISS( ) — конструктор, в котором вызывается Min\_Prime\_Nums( ) после чего значение square\_size меняется и на основе которой выделяется память для optimal\_square и cur\_square. После чего происходит первое заполнение optimal\_square 3 самыми большими квадратами которые помогают ускорить процесс вычисления лучшего ответа.
* Min\_Prime\_Nums( ) — метод, находящий меньшее простое число, которое записывется в square\_size, а также меняет proportion, чтобы при выводе вернуть изначальный размер введенный пользователем.
* Fill(int Y, int X, int size) — метод, который добавляет новые квадраты в матрицу optimal\_square, меняя их значение в координатах y и x на -1, после инкрементирует cur\_count.
* Backtracking (int start\_y, int start\_x, int end) — основной процесс подбора квадратов с различными размерами. Описан в разделе Выполнение работы.
* Controlled\_Fill(int Y, int X, int size) — метод, который проверяет возможность добавления квадрата с координатами Y X и размером size в optimal\_square. В случае если это возможно возвращается true и вызывается Fill а в инном случае просто возвращает false.
* Next\_backtracking( int start\_y, int start\_x, int end, int i) — метод, который помогает методу Backtracking в поиске следующего квадрата для оптимального запуска Backtracking.
* Remove( int Y, int X, int size) — метод, который проходит по координатам y и x при встрече числа, отличающегося от 0, меняет его на 0 и таким образом удаляет квадраты из optimal\_square, после чего декрементирует cur\_count.
* result\_output( ) — метод, который печатает лучшее решение в требуемом формате.

**Выводы**

В ходе данной работы были изучены алгоритм поиска с возвратом, метод ветвей и границ. Также придуманы способы оптимизации решение задачи поиска минимального количество квадратов вмещаемых в квадрат размером n.

Способы оптимизации:

* Сводить квадрат размера n к квадрату с размером наименьшего простого числа позволяет алгоритму работать быстрее из-за уменьшенной площади работы.
* Задавать начальные квадраты размером (n+1)/2 и 2 квадрата размером (n-1)/2 также помогло уменьшить площадь работы алгоритма ускоряя его работу.

Алгоритм успешно выполняется для 2 ≤ N ≤ 40 меньше чем 9 секунд.