# 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. 1384 | Бобков В. Д. |
| Преподаватель | Шевелева А.М. |

Санкт-Петербург 2023

# 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 (см. рис. 1).

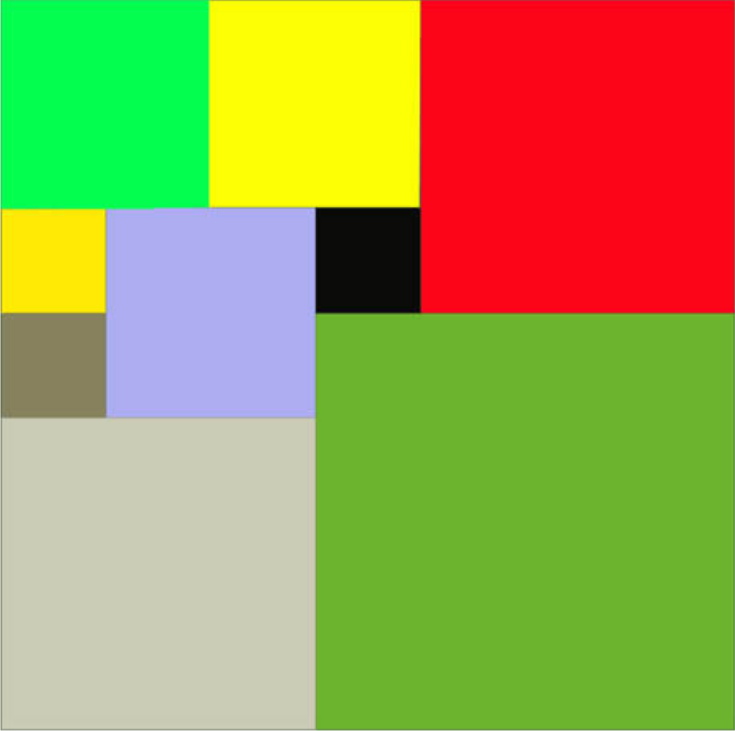


Рисунок 1 - Пример столешницы 7×7

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\*N. Далее должны идти K строк, каждая из которых должна содержать три целых числа x, y и w, задающие координаты левого верхнего угла 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 начала был создан класс *Square*, хранящий координаты левого верхнего угла и размер квадрата, который служит частью стола. Затем были реализованы следующие методы и поля данного класса: конструктор, перегруженный метод *\_\_str\_\_*, метод *change\_coord*, который меняет значения полей *x,y,w* объекта класса.

Функция *simplification* служит для нахождения наименьшего простого множителя, чтоб вместо заполнения большого стола заполнялось маленькое, а затем результат преобразовывался для большего размера стола.

Функция *step\_back* служит как раз для возвращения поля к начальным размерам стола.

Переменная *k* хранит в себе размер уменьшенного поля.

Переменная *record* хранит в себе количество квадратов в наилучшей сборке.

Массив *best\_case* хранит в себе список рекордного количества квадратов.

Функция *is\_inside* определяет, находится ли точка с координатами *x y* внутри одного из квадратов.

Функция constructing есть реализация поиска с возвратом. В ней перебираются все точки с координатами *x, y*, проверяется принадлежность каждой одному из квадратов, затем находится максимальная возможная длина стороны квадрата, который можно вставить, затем в цикле перебираются все варианты вплоть до самого маленького квадрата, который можно вставить. После данных операций идет проверка на то, заполнена ли столешница, если да, то количество квадратов в сборке сверяется с рекордно-минимальным, если найдена лучшая сборка, данные о рекорде обновляются.

Были применены следующие оптимизации:

1. Для каждого числа N искался наименьший простой его множитель, затем поле заполнялось как раз для наименьшего простого множителя данного числа. Получается, что так происходит намного меньше итераций, нежели чем при заполнении изначального поля. После выполнения функции constructing с помощью функции step\_back происходит возвращение к прошлому масштабу
2. Если стол не является заполненным и количество квадратов в нём равняется количеству квадратов в прошлом решении, уменьшенном на единицу, нахождение ответа можно прекратить, так как при следующем рекурсивном вызове функции необходимо будет поставить на столешницу ещё один квадрат, после чего их количество будет не менее количества квадратов в решении.
3. В самом начале ставится 3 квадрата: в верхний левый угол со стороной (n+1)//2 и два по бокам от него со сторонами n//2. Таким образом, нужно заполнять не квадрат со стороной n, а квадрат со стороной n/2, что значительно уменьшает количество итераций.

## **Тестирование**.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
| 1 | 7 | 7  9  1 1 4  1 5 3  5 1 3  4 5 2  4 7 1  5 4 1  5 7 1  6 4 2  6 6 2 | Верно |
| 2 | 20 | 20  4  1 1 10  1 11 10  11 1 10  11 11 10 | Верно |

Продолжение таблицы 1

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
| 3 | 29 | 29  14  1 1 15  1 16 14  16 1 14  15 16 2  15 18 5  15 23 7  16 15 1  17 15 3  20 15 3  20 18 3  20 21 2  22 21 1  22 22 8  23 15 7 | Верно |

**Выводы**

В ходе выполнения работы был изучен, реализован и применён на практике метод решения задач на тему «Поиск с возвратом». Данный алгоритм позволил найти все решения поставленной задачи, если они существуют. Для ускорения метода вычисления были организованы таким образом, чтобы как можно раньше выявлять заведомо неподходящие варианты. Это позволило значительно уменьшить время нахождения решения.

# ПРИЛОЖЕНИЕ А

# ИСХОДНЫЙ КОД ПРОГРАММЫ

# Файл main.py:

class Square: # класс для квадрата

    def \_\_init\_\_(*self*,*x*,*y*,*w*): # конструктор класса

*self*.x = *x*#координаты верхнего

*self*.y = *y*#левого угла

*self*.w = *w*#длина стороны

    def \_\_str\_\_(*self*):# перегруженный метод str

        return str(*self*.x + 1) +' ' + str(*self*.y + 1) + ' '+ str(*self*.w)

    def change\_coord(*self*,*x*,*y*,*w*):# метод для возвращения размера квадратов и их координат к изначальному масштабу

*self*.x = *x*

*self*.y = *y*

*self*.w = *w*

def simplification(*N*): # получаем наименьший простой множитель

    for i in range(2,6):

        if *N* % i == 0:

            return i

    return n

n = int(input())

k = simplification(n)# получаем наименьший простой множитель, чтоб было проще и быстрее заполнять

record = 2\*n + 1

best\_case = [] # рекордные значения сборки

def step\_back(*squares*,*k*): # возврат к первоначальному масштабу стола

    for s in *squares*:

        s.change\_coord(s.x \**k*, s.y \* *k*, s.w \* *k*)

def is\_inside(*squares*,*x*,*y*): # проверка на нахождение точки внутри какого-либо из квадратов

  for square in *squares*:

    if (*x* >= square.x and *x* < square.x + square.w and *y* >= square.y

        and *y* < square.y + square.w):

      return 1

  return 0

def constructing(*squares*, *S*, *xmin*,*ymin*): # главная функция

    global record

    for x in  range(*xmin*,k):

        for y in range(*ymin*,k): # двигаемся сверху-вниз

            if not is\_inside(*squares*,x,y):

                new\_len = min(k - x, k - y) # наибольший квадрат, которые можно вставить, учитывая только края стола

                for square in *squares*:

                    if (square.x + square.w > x and square.y > y):

                        new\_len = min(new\_len, square.y - y) # учитываем то, что снизу от точки могут быть еще квадраты от прошлых проходок по столу

                for i in range(new\_len,0,-1):

                    s= Square(x,y,i)

                    tmp = *squares*.copy()

                    tmp.append(s)

                    if *S* + s.w \*\* 2 == k \*\* 2: # новое решение

                        if len(tmp) < record: # проверка на успешность решения

                            record = len(tmp)

                            best\_case[:] = tmp.copy()

                    else:

                        if len(tmp) < record: # идем дальше по дереву, если в данном решении есть смысл

                            constructing(tmp,*S* + s.w \*\* 2, x, y+i)

                        else:

                            return

                return

*ymin* =  k//2 # после первой проходки вниз ставим новый ymin

        # потому что изначально из первого большего квадрата вниз торчал уголок,

        # из-за которого не было смысла начинать оттуда, так как клетка была занята

def main():

    squares = []

    squares.append(Square(0,0,(k+1)//2))

    squares.append(Square(0,(k+1)//2, k//2))

    squares.append(Square((k+1)//2,0, k//2)) # сразу же заполняем 3/4 квадрата наиболее оптимальным путем

    constructing(squares, ((k+1)//2) \*\* 2 + 2\* ((k//2) \*\* 2),k//2, (k+1)//2)

    if k!=n:

        step\_back(best\_case,n//k)

    print(record)

    for s in best\_case:

        print(s)

main()