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. 9383 _____ Звега А.Р.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Изучение и практическое освоение алгоритма поиска с возвратом.

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.

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).

Задание (Вариант 4р).

Рекурсивный бэктрекинг. Расширение задачи на прямоугольные поля, ребра квадратов меньше ребер поля. Подсчет количества вариантов покрытия минимальным числом квадратов.

Выполнение работы.

Для реализации программы был выбран рекурсивный алгоритм поиска с возвратом. Полное описание алгоритма:

1) Ввод двух целых чисел width и height (2 <= width, height <= 20).
2) Выбирается начальный квадрат.
3) Если есть место в прямоугольнике вставляется квадрат, повторяется пока место будет.
4) Если места нет, количество квадратов сравнивается с текущим минимумом, то что меньше становится минимумом.
5) Программа выходит из функции, пока не найдет место, где можно вставить квадрат меньшей длинны.
6) Если такое место есть, то программа переходит на шаг 3.
7) Если такого места нет, то возвращается найденный ответ

Детали реализации.

Описание функций:

find_paving(width, height, curent_min, box, min, ans = list()) – рекурсивная функция, которая решает задачу. В качестве аргументов функция принимает: ширину и высоту поля, текущий минимум, начальный квадрат, количество уже вставленных квадратов, список вставленных квадратов. Возвращает список который содержит, число квадратов и число решений.

check_box(width, height , box, ans) – функция проверяет, можно ли вставить квадрат со стронной box, в список квадратов ans, если поле имеет высоту width и ширину height, если место есть возвращает координаты левого верхнего угла и высоту квадрата, если места нет возвращает пустой массив.

start(width, height) – определяет начальный квадрат и начальный минимум.

Тест 1 – проверяет работу функции start, правильно ли она считывает данные. Тест 2 – проверяет работу функции find_paving, корректный ли она выдает ответ.

Тест 3 – проверяет работу функции check_box, правильно ли она располагает квадрат в области.

Демонстрация работы программы показана на Рисунках 1 и 2.

Результаты тестирования представлены в Таблице 1.

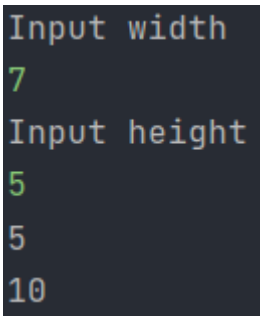

Рисунок 1. Результат работы программы при width = 7, height = 5

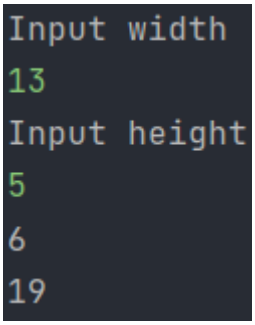

Рисунок 2. Результат работы программы при width = 13, height = 5

Таблица 1. Результаты тестирования

Ввод	Вывод
5 7	5 10
10 13	7 45
11 4	6 8
12 19	7 95

Анализ работы алгоритма.

Алгоритм поиска с возвратом значительно ускоряет работу программы, так как не нужно проверять все расположения квадратов, а только те которые не превышают текущее минимальное решение.

При тестировании программы можно увидеть, что при увеличении размеров поля, время работы алгоритма увеличивается экспоненциально. Так же, алгоритм работает значительно медленнее для полей в составе которых(длинна или высота) простые числа, так как у них значительно больше решений, а в полях у которых стороны кратны 2 решении очень мало.

Оптимизация при простых числах заключается в отсечении вершин, таким образом сокращается диапазон перебора.

Выводы.

При выполнении работы был освоен рекурсивный алгоритм поиска с возвратом. Была реализована программа, которая позволяет определить минимальное количество квадратов, из которых можно составить прямоугольник.

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД.

Название файла: rect_lab1.py

```
import time

def check_box(width, height , box, ans):
    for y in range(height-box+1):
        x = 0
        while x < width-box+1:
            num = len(ans)
            if ans:
                for i in ans:
                    if i[0] <= y and i[1] <= x:
                        if i[1]+i[2]-1 >= x and i[0]+i[2]-1 >= y:
                            x += i[2] - 1
                            break
                    elif i[0] <= y and i[1] >= x:
                        if x+box-1 >= i[1] and i[0]+i[2]-1 >= y:
                            x += i[2] - 1
                            break
                    elif i[0] >= y and i[1] <= x:
                        if y+box-1 >= i[0] and i[1]+i[2]-1 >= x:
                            x += i[2] - 1
                            break
                    else:
                        if y + box - 1 >= i[0] and x+box-1 >= i[1]:
                            x += i[2] - 1
                            break
                    num -= 1
                if not num:
                    return [y, x, box]
            else:
                return [y, x, box]
            x += 1
    return False
```

```
def find_paving(width, height, curent_min, box, min, ans = list()):
    global ans_min
    global count
    if ans_min:
        if len(ans) > len(ans_min):
            return ans_min
    if len(ans) > curent_min:
        return ans_min
    fast = 0
    skip_box = 0

    for i in range(box - skip_box, 0+fast, -1):
        check = check_box(width, height, i, ans)
        if check:
            find_paving(width, height, curent_min, box, min + 1, ans +
list([check]))
        elif i == 1:
            if ans_min:
                if len(ans) < len(ans_min):
                    ans_min = ans
                    return ans
                elif len(ans) == len(ans_min):
                    count += 1
                return ans_min
            else:
                ans_min = ans
        else:
            continue
    return [len(ans_min), count]


def start(width, height):
    if width < height:
        buf = height
        height = width
        width = buf

    curent_min = width * height
    rect = True
```

```
    if width == height:
        curent_min = 13
        rect = False

    if rect:
        box = height
    elif not width % 2:
        box = int(width / 2)
    elif not width % 3:
        box = int(width / 3 * 2)
    else:
        box = int(width / 2)+1

    if (width >= 2 and height >= 2) and (height <= 20 and width <= 20):
        return find_paving(width, height, curent_min, box, 0)
    else:
        print('2 <= width,height <= 20')


if __name__ == '__main__':
    width = int(input())
    height = int(input())
    ans_min = list()
    count = 1
    start(width, height)
    print(len(ans_min))
    print(count)
```