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 _____ Соседков К.С.

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 4и).

Итеративный 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 итеративный алгоритм поиска с возвратом. Полное описание алгоритма приведено ниже:

0) Ввод двух целых чисел N и M. N – ширина прямоугольника, M – высота (2 <= N,M <= 20).

1) Поиск свободного места в прямоугольнике для вставки в него квадрата.

2) Если в прямоугольнике есть свободное место, вставить в него квадрат максимальной ширины и перейти к шагу 1.

3) Если места в прямоугольнике нет, значит текущий набор квадратов является решением. Сравнить минимальное решение с текущим. Удалить из решения последний вставленный квадрат.

4) Если ширина последнего вставленного квадрата равна единице, удалить с конца все квадраты ширины 1 пока не встретится квадрат с шириной больше чем 1.

5) Если ширина больше 1, уменьшить ширину квадрата на единицу и перейти к шагу 1.

Детали реализации.

Для хранения прямоугольников в памяти был реализован класс Rect. Класс Rect содержит четыре поля – координаты левого верхнего угла(x,y), и размер(width, height).

Описание функций:

solve(W,H) – основная функция решающая данную задачу. В качестве аргументов функция принимает два целых числа - высоту и ширину прямоугольника. Возвращает кортеж содержащий два элемента – массив квадратов минимальной длинны и количество решений с минимальным числом квадратов.

create_square(free_rect, square_size) – функция вписывает квадрат размером square_size в прямоугольник free_rect в правый нижний угол и возвращает полученный квадрат.

find_free_rect(squares, y_min) – функция определяет осталось ли свободное место в основном прямоугольнике, если место есть – возвращает свободный прямоугольник.

Демонстрация работы программы показана на Рисунках 1 и 2.
Результаты тестирования представлены в Таблице 1.

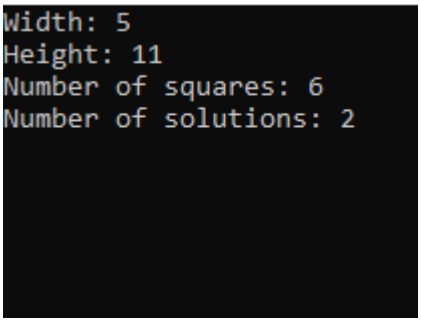

Рисунок 1. Результат работы программы при N=5, M=11

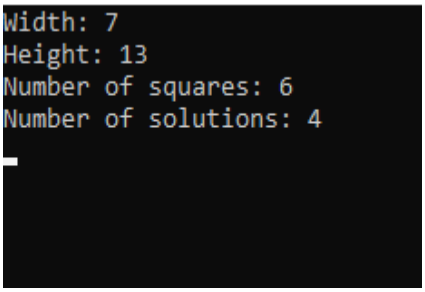

Рисунок 2. Результат работы программы при N=7, M=13

Таблица 1. Результаты тестирования

Ввод	Вывод
5 11	Number of squares: 6 Number of solutions: 2
7 13	Number of squares: 6 Number of solutions: 4
11 9	Number of squares: 7 Number of solutions: 5
-23 2	2 <= N,M <= 20
10 10	Number of squares: 4 Number of solutions: 1
4 17	Number of squares: 8 Number of solutions: 1

Выводы.

При выполнении работы был освоен итеративный и рекурсивный алгоритм поиска с возвратом. С помощью данного алгоритма была реализована программа, которая позволяет определить минимальное количество квадратов из которых можно составить один прямоугольник размера NxM.

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД.

Название файла: lab1.py

```
class Rect:
	x = 0
	y = 0
	width = 0
	height = 0

	def __init__(self, x, y, width, height):
		self.x = x
		self.y = y
		self.width = width
		self.height = height

	def __repr__(self):
		return str(self.x) + "," + str(self.y) + "," + str(self.width) + "," + str(self.height)




def create_square(free_rect, square_size):
	right_down_corner = (free_rect.x+free_rect.width, free_rect.y+free_rect.height)
	new_x = right_down_corner[0]-square_size
	new_y = right_down_corner[1]-square_size
	return Rect(new_x, new_y, square_size, square_size)




def find_free_rect(squares, y_min):
	max_value = max(y_min)
	max_index = y_min.index(max_value)

	count = 0

	for i in range(max_index, len(y_min)):
```

```
            if y_min[i] != max_value:
                break
            count += 1

    return Rect(max_index, 0, count, max_value)




def update(squares, y_min):
    while squares and squares[-1].width == 1:
        new_square = squares[-1]
        squares.pop()
        y_min = y_min[0:new_square.x] + [new_square.y+new_square.height]*(new_square.width)
+ y_min[new_square.x+new_square.width:]


    if not squares: return ([],[])

    new_square = squares[-1]
    squares.pop()
    y_min = y_min[0:new_square.x] + [new_square.y+new_square.height]*(new_square.width) +
y_min[new_square.x+new_square.width:]
    free_rectangle = find_free_rect(squares, y_min)

    new_square = create_square(free_rectangle, new_square.width-1)

    squares.append(new_square)
    y_min = y_min[0:new_square.x] + [new_square.y]*(new_square.width) +
y_min[new_square.x+new_square.width:]

    return (squares, y_min)
```

```
def solve(W,H):
    y_min = [H]*(W)
    squares = []
    min_solution = []
    count = 1

    while True:
        free_rectangle = find_free_rect(squares, y_min)

        if min_solution and len(squares) > len(min_solution):
            squares, y_min = update(squares, y_min)
            if not squares: break

        elif free_rectangle.height > 0:
            square_width = min(free_rectangle.width, free_rectangle.height)
            if not squares:
                square_width -= 1

            new_square = create_square(free_rectangle, square_width)
            squares.append(new_square)
            y_min = y_min[0:new_square.x] + [new_square.y]*(new_square.width) +
y_min[new_square.x+new_square.width:]

        else:
            if not min_solution:
                min_solution = squares[:]
            elif len(min_solution) > len(squares):
                count = 1
                min_solution = squares[:]
            elif len(min_solution) == len(squares):
                count += 1

            squares,        y_min = update(squares, y_min)
            if not squares: break

    return (min_solution, count)
```

```
if __name__ == '__main__':
        N = int(input())
        M = int(input())

        if N >= 2 and M >= 2:
                solution = solve(N, M)
                print('Number of squares:', len(solution[0]))
                print('Number of solutions:', solution[1])
                input()
        else:
                print('2 <= N,M <= 20')
                input()
```