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
Вариант: 3р

Студент гр. 3388 _____ Кулач Д.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2025

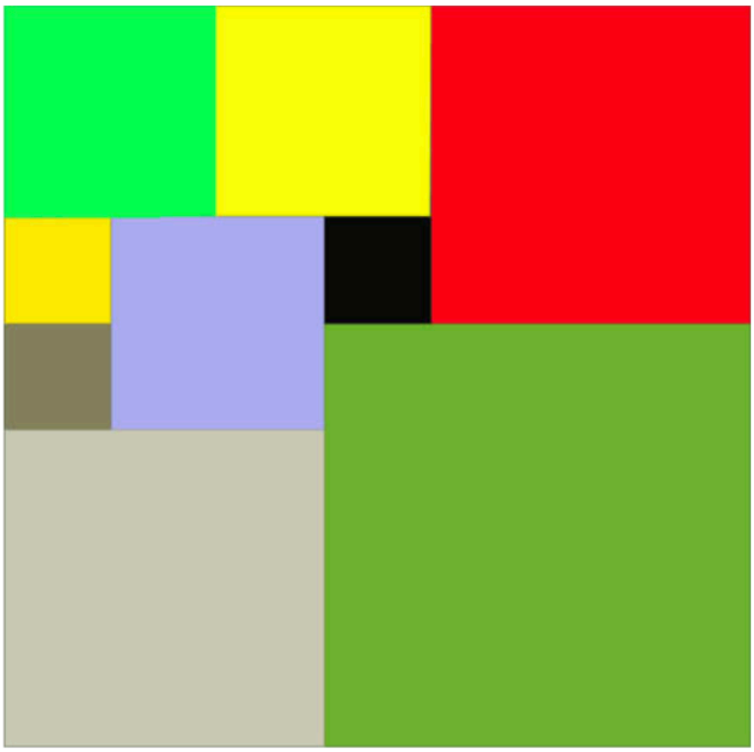
Цель работы:

Изучить принцип работы алгоритма поиска с возвратом. Решить с его помощью задачу. Также провести исследование зависимости количества итераций от стороны квадрата.

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 $(2 \leq N \leq 20)$.

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

112

132

311

411

322

513

444

153

341

Реализация

Описание алгоритма:

Для решения поставленной задачи был использован рекурсивный бэктрекинг (рекурсивный поиск с возвратом). Так, после ввода стороны генерируется набор возможных высот для начального размещения крупных квадратов. Этот набор зависит от размера входного значения. Для каждой возможной высоты из набора выполняет следующие действия:

1. Инициализирует начальную диаграмму высот прямоугольника с учётом размещения крупных начальных блоков.
2. Вызывает рекурсивную функцию поиска оптимального решения.
3. Обновляет лучшее решение, если найденное решение лучше текущего лучшего решения.

Описание функций и структур:

- `squares_masks` – словарь, в котором для каждого возможного квадрата `(x, y, s)` (левый верхний угол и размер стороны) хранится битовая маска клеток, которые он покрывает. Это ускоряет проверку наложений.
- `find_first_free(board, start_idx)` – возвращает индекс первой свободной (непокрытой) клетки в битовой маске `board`, начиная с позиции `start_idx`.
- `print_solution_matrix(n, result)` – формирует и печатает матрицу с номерами квадратов, покрывающих соответствующие ячейки. Используется для отладки и визуализации результата.
- `search(board, current_count, start_idx, free_count)` – основная рекурсивная функция поиска:
    - `board` – текущая битовая маска занятых клеток;
    - `current_count` – текущее количество размещённых квадратов;

- `start_idx` – индекс для ускоренного поиска свободных клеток;
- `free_count` – число оставшихся незанятых клеток.

• `best_count` и `best_solution` – глобальные переменные, хранящие текущее лучшее (наименьшее) решение.

• `dp` – словарь, реализующий динамическое программирование: хранит минимальное число квадратов, использованное для каждой маски `board`, чтобы избежать повторной обработки одних и тех же конфигураций.

• `solution` – стек текущего решения. После завершения рекурсивного вызова последний квадрат из стека удаляется (обратный шаг бэктрекинга).

Способ хранения частичных решений:

• `solution` – список, хранящий кортежи (x, y, s) для всех квадратов текущей конфигурации;

• `best_solution[0]` – список, содержащий квадраты лучшего (оптимального) покрытия;

• Маска `board` – битовое представление занятой/свободной области размером N×N.

Алгоритмы оптимизации:

• Битовые маски – обеспечивают быстрые операции проверки пересечений и обновления состояния области;

• Оценка нижней границы (lower_bound) – минимальное количество квадратов, необходимое для покрытия оставшейся области. Если текущее решение + оценка ≥ лучшее найденное – ветвь обрезается;

• Мемоизация (dp) – исключает повторный перебор одинаковых конфигураций;

• Ограничение максимального размера квадрата – максимальный размер ограничивается (N-1), чтобы избежать лишних проверок;

• Ранний выход – если уже текущий путь не может дать более оптимального решения, выполнение прерывается.

Оценка сложности алгоритма:

Основной идеей алгоритма является рекурсия, соответственно количество возможных переборов будет расти, как степенная функция. Ввиду использования оптимизаций, сложность алгоритма уменьшается на некоторую константу, но в худших всё ещё приближается к экспоненциальной сложности (O(e^n)), где n-сторона квадрата.

Тестирование

Таблица 1. Тестирование.

Входные данные	Выходные данные
2	4 1 1 1 2 2 1 1 2 1 2 1 1
3	6 1 2 1 2 1 1 1 1 1 2 2 2 1 3 1 3 1 1
11	11 5 6 1 4 6 1 4 3 3 1 4 3 5 1 2 4 2 1 4 1 1 1 1 3 6 6 6 1 7 5 7 1 5
15	6 1 6 5 6 1 5 1 1 5 6 6 10 1 11 5 11 1 5

19	13 9 10 1 8 10 1 8 7 3 4 7 4 1 8 3 5 1 6 4 6 1 4 5 1 1 5 3 1 1 4 10 10 10 1 11 9 11 1 9

Иследование

Также в ходе лабораторной работы было проведено исследование зависимости количества итераций от стороны квадрата. В ходе исследования получились следующие результаты(рис. 1 и табл. 2).

Таблица 2. Зависимость количества итераций от стороны квадрата.

Сторона квадрата	Количество итераций
3	11
4	21
5	71
6	48
7	308
9	212
11	5361
12	181
13	14890
15	971
17	2754

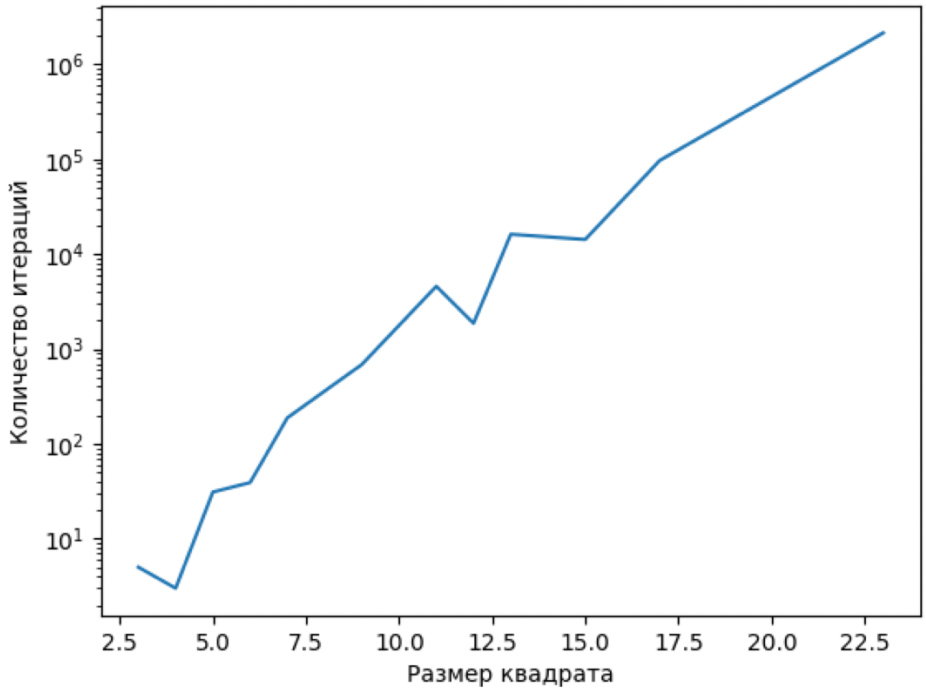


Рис. 1. Зависимость количества итераций от стороны квадрата

Построим логарифмический график зависимости количества итераций от стороны квадрата. Не сложно заметить, что значения в простых числах образуют прямую, что свидетельствует о экспоненциальной зависимости.

Вывод

В ходе лабораторной работы была написана программа с использованием алгоритма бэктрекинга. Также было проведено тестирование на различных входных данных. По результатом исследования можно заключить, что зависимость числа операций от размера поля экспоненциальна.

Исходный код программы см. в ПРИЛОЖЕНИИ А.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД ПРОГРАММЫ

main.py

```
def find_min_squares_with_debug(N, DEBUG=True):
    import sys
    sys.setrecursionlimit(10000)

    total_cells = N * N
    full_mask = (1 << total_cells) - 1

    squares_masks = {}
    for y in range(N):
        for x in range(N):
            max_side = min(N - x, N - y)
            for s in range(1, max_side + 1):
                if s == N:
                    continue
                mask = 0
                for r in range(y, y + s):
                    for c in range(x, x + s):
                        mask |= (1 << (r * N + c))
                squares_masks[(x, y, s)] = mask

    def find_first_free(board, start_idx):
        for i in range(start_idx, total_cells):
            if ((board >> i) & 1) == 0:
                return i
        return -1

    best_count = [float('inf')]
    best_solution = [[]]
    dp = {}
    solution = []
    iteration_counter = [0]

    def print_solution_matrix(n, result):
        print("\nИтоговая матрица решения")
        matrix = [[0] * n for _ in range(n)]
        num = 1
        for (x, y, s) in result:
            for i in range(s):
                for j in range(s):
                    matrix[y - 1 + i][x - 1 + j] = num
            num += 1

        for row in matrix:
            print(" ".join(f"{cell:2}" for cell in row))

    def search(board, current_count, start_idx, free_count):
        iteration_counter[0] += 1
        iter_num = iteration_counter[0]

        if DEBUG:
            print(f"\nИтерация #{iter_num}:")
            print(f"Текущий стек ({len(solution)} квадратов):")
            for sq in solution:
                print(f"\tКвадрат: ({sq[0]}, {sq[1]}) размер {sq[2]}")
```

```
        if current_count >= best_count[0]:
            if DEBUG:
                print("\tОбрезка ветви: текущих квадратов уже больше
оптимума")
            return
        if board == full_mask:
            if current_count < best_count[0]:
                best_count[0] = current_count
                best_solution[0] = solution.copy()
                if DEBUG:
                    print(f"\tНайден новый лучший результат:
{current_count} квадратов")
            return
        if board in dp and dp[board] <= current_count:
            return
        dp[board] = current_count

        idx = find_first_free(board, start_idx)
        if idx == -1:
            return
        x = idx % N
        y = idx // N

        max_possible = min(N - x, N - y)
        if max_possible == N:
            max_possible = N - 1
        if max_possible <= 0:
            max_possible = 1
        max_area = max_possible * max_possible
        lower_bound = (free_count + max_area - 1) // max_area
        if current_count + lower_bound >= best_count[0]:
            if DEBUG:
                print("\tОбрезка ветви: оценка нижней границы >=
текущего оптимума")
            return

        max_side = min(N - x, N - y)
        if max_side == N:
            max_side = N - 1

        for s in range(max_side, 0, -1):
            mask = squares_masks.get((x, y, s))
            if mask is None:
                continue
            if board & mask == 0:
                solution.append((x + 1, y + 1, s))
                if DEBUG:
                    print(f"\tДобавляем квадрат: ({x + 1}, {y + 1})
размер {s}")
                search(board | mask, current_count + 1, idx + 1,
free_count - s * s)
                if DEBUG:
                    print(f"\tУбираем квадрат: ({x + 1}, {y + 1})
размер {s}")
                solution.pop()

    search(0, 0, 0, total_cells)

    return best_count[0], best_solution[0], iteration_counter[0],
```

```
print_solution_matrix


if __name__ == "__main__":
    N = int(input("Размер столешницы N: "))
    if 2 <= N < 20:
        count, squares, iterations, print_matrix_fn =
find_min_squares_with_debug(N, DEBUG=True)
        print(f"\nМинимальное количество квадратов: {count}")
        for x, y, s in squares:
            print(f"{x} {y} {s}")
        print(f"\nКоличество итераций: {iterations}")
        print_matrix_fn(N, squares)
    else:
        print("Недопустимый размер. Введите число от 2 до 19.")
```