МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Построение и анализ алгоритмов»

ТЕМА: «ПОИСК С ВОЗВРАТОМ»

ВАРИАНТ 4Р

Гусакова К.А.

Студент гр. 3388 _____

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2025

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

У Вовы много квадратных обрезков доски. Их стороны (размер) изменяются от 11 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 на вариант:
Рекурсивный бэктрекинг. Расширение задачи на прямоугольные

поля, рёбра квадратов меньше рёбер поля. Подсчёт количества вариантов покрытия минимальным числом квадратов.

Описание алгоритма для решения задачи

С консоли вводятся n и m. Если n > m, то они меняются местами для упрощения. Далее создаётся массив heights[n], где heights отражает текущую "высоту" закрашивания в каждой колонке прямоугольника (от 0 до m).

Заводится переменная bestCnt, равная n * m(т.к. максимальное число квадратов – n*m квадратов размером 1, покрывающих весь прямоугольник) – это максимальное возможное количество квадратов.

Начинается рекурсивный перебор с глубины 0(DFS level, который показывает откуда мы идем и куда мы откатываемся для перебора).

Поиск свободной области

На каждом шаге мы находим колонку с минимальной высотой minHeight – это наиболее "свободное" место. От этой колонки left вправо ищем максимальное количество подряд идущих колонок с этой же высотой – чтобы определить максимально возможный размер квадрата, который можно туда поставить.

left – индекс самой низкой колонки.

right – крайний правый индекс, пока высоты равны minHeight.

right - left – ширина свободного пространства, т.е. максимально возможная ширина квадрата, который туда можно вставить.

Вставка квадрата(или откат)

Для всех возможных размеров квадратов size (от максимального до 1) мы проверяем, помещается ли квадрат в текущую "впадину". Если да – добавляем квадрат, обновляем heights[], увеличивая значения высоты на size. Сохраняем квадрат в текущем пути currentPath[]. Вызываем рекурсивно dfs(...), увеличивая уровень глубины (cnt).

После возврата из рекурсии – откатываем изменения на шаг назад(что и является бэктрекингом): удаляем квадрат из currentPath[], уменьшаем соответствующие значения в heights[].

Условие завершения

Если все значения в heights[] равны m, значит весь прямоугольник покрыт. Если текущее количество квадратов cnt меньше bestCnt, то обновляется bestCnt и запоминается текущий путь как наилучшее решение. Если текущее количество квадратов равно bestCnt, то увеличивается счётчик способов (countWays).

Оценка сложности алгоритма по времени и памяти

Временная сложность dfs:

1. if (cnt > bestCnt) return; - сложность O(1).
2. if (minHeight == m) {

if (cnt < bestCnt)… - проверка все ли покрыто, сложность O(n*m) в худшем случае.

3.

int maxSize = right - left;

if (m - minHeight < maxSize) maxSize = m - minHeight; - Вычисление maxSize O(1)

4.

for (int size = maxSize; size >= 1; size--) {

...

} - Основной цикл по размерам квадратов, где максимум - min(n, m) итераций, сложность O(min(n, m))

5.

Мы ищем свободное место (O(n)):

```
int left = 0, minHeight = m;
for (int i = 0; i < n; i++) {
   if (heights[i] < minHeight) {
      minHeight = heights[i];
      left = i;
   }
}
```

Пробуем разместить начиная с самого большого и спускаясь до 1, их может быть до O(min(n, m)), рекурсивно продолжаем с уменьшенным числом свободных клеток.

Максимальная глубина рекурсии — это n*m, когда площадь прямоугольника полностью покрыта.

Максимальная временная сложность:

$O((\min(n, m))^{(n * m)})$

где:

n * m – площадь прямоугольника (максимальное число квадратов),

k – число возможных размеров квадратов (в худшем случае до min(n, m)),

S - n * m

Итог k^S – сложность алгоритма (экспоненциальная сложность).

Пространственная сложность:

Сколько памяти используется в пике:

heights[MAX_N] – O(n) (до 20)

currentPath[MAX_N * MAX_N] – путь текущего варианта, максимум O(n * m)

bestPath[MAX_N * MAX_N] – лучший путь, тоже O(n * m)

Рекурсивный вызов dfs() – максимум O(n * m) вызовов

O(n*m)

в памяти, где n * m — площадь прямоугольника.

Описание способа хранения частичных решений.

Square currentPath[MAX_N * MAX_N]; массив, который хранит текущую последовательность поставленных квадратов.

Перед рекурсивным вызовом: текущий квадрат добавляется в currentPath. При нахождении наилучшего пути, частичное решение становится лучшим решением.

Использованные оптимизации.

1. Мы отсекаем решение хуже, чем уже найденное, тем самым не продолжая перебор этого варианта.
if (cnt > bestCnt) return;

2. Перебор начинается с больших возможных квадратов, дабы быстрее заполнить площадь(больший квадрат ведет к меньшей площади и возможности поставить еще квадраты), что ведет к более эффективному заполнению.
for (int size = maxSize; size >= 1; size--));

Рекурсивная функция dfs.

Сигнатура	void dfs(int n, int m, int heights[], int cnt, int& bestCnt, Square currentPath[], Square bestPath[], int& currentLen);		
Назначение	Ищет оптимальное разбиение прямоугольника n × m на квадраты, используя метод глубокого поиска с возвратом		
Аргументы	n	int	Ширина прямоугольника
	m	int	Высота прямоугольника
	heights[]	int[]	Массив высот для каждой колонки прямоугольника
	cnt	int	Текущее количество использованных квадратов
	bestCnt	int&	Ссылка на текущее минимальное количество квадратов (лучшее решение)

	currentPath[]	Square[]	Массив квадратов, использованных на текущем шаге (текущее решение)
	bestPath[]	Square[]	Массив квадратов, соответствующих лучшему решению
	currentLen	int&	Текущий размер пути currentPath (сколько квадратов в нём)
Возвращаемое значение	Не возвращает результат(void). Все изменения происходят через ссылки и глобальные переменные (bestCnt, bestPath, countWays)		

Тестирование. Показ граничных случаев алгоритма.

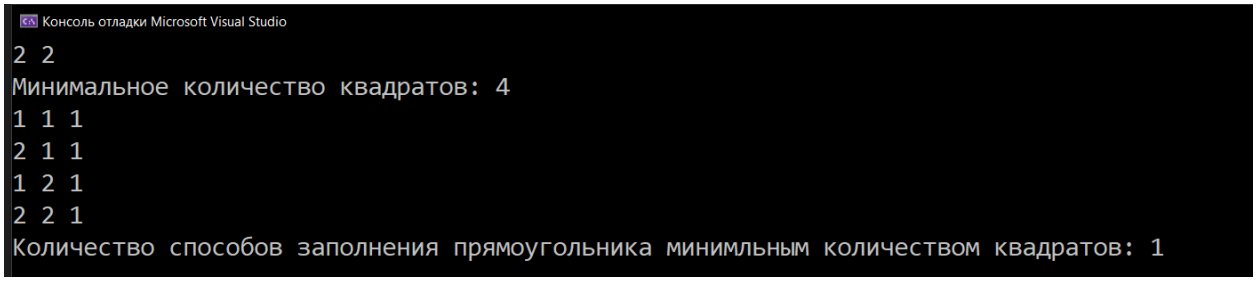

Рисунок 1 – минимальный квадрат.

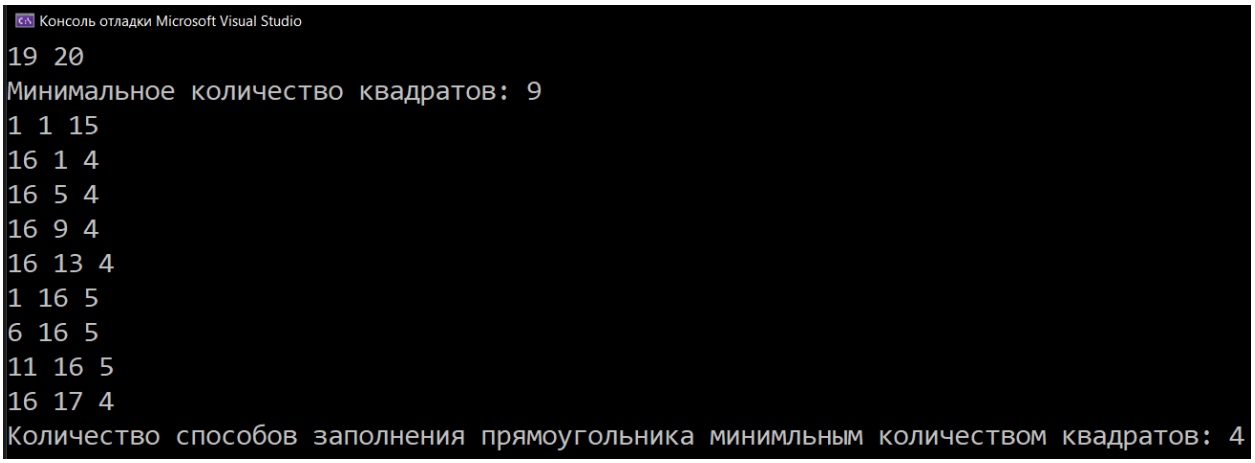

Рисунок 2 – прямоугольник близкий к максимальному значению.

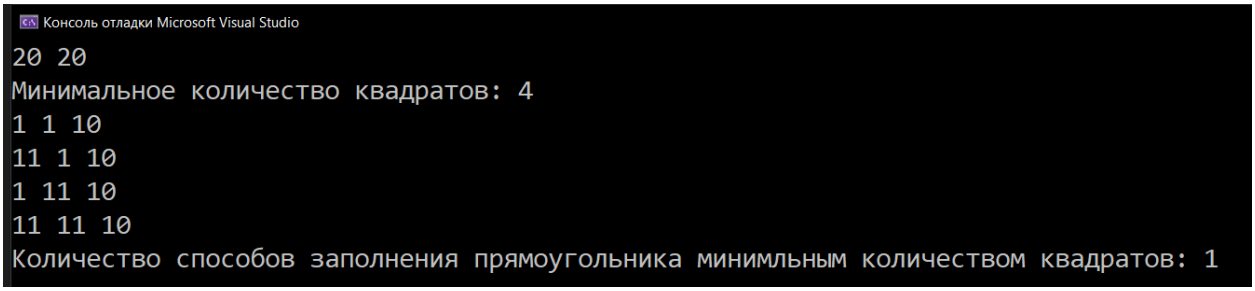

Рисунок 3 – максимальный квадрат.

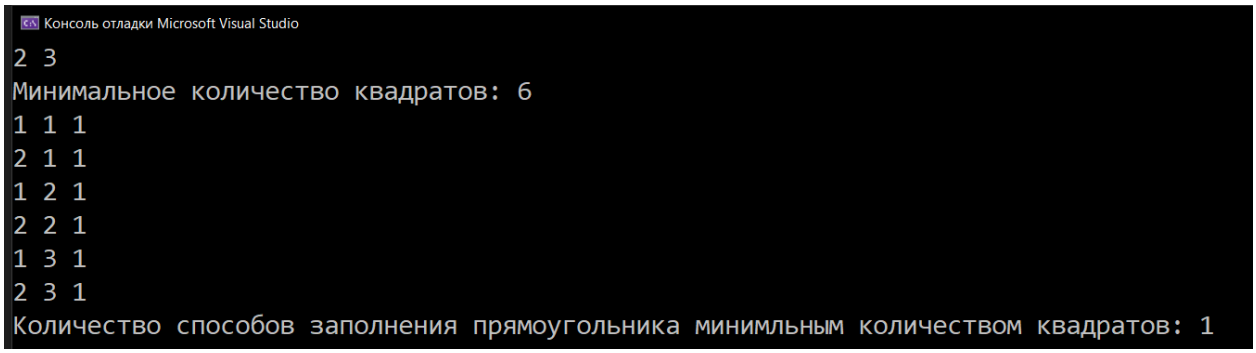

Рисунок 4 – минимальный прямоугольник

ВЫВОД

В ходе выполнения лабораторной работы был реализован алгоритм покрытия прямоугольника квадратами с минимальным числом элементов, основанный на рекурсивном бэктрекинге с оптимизациями.

ПРИЛОЖЕНИЕ. ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
using namespace std;

const int MAX_N = 20;

struct Square {
    int x = 0, y = 0, w = 0;
};

/*class Solution {
public:
```

```
    int tilingRectangle(int n, int m) {
        if (n > m) swap(n, m);

        int heights[MAX_N] = { 0 };
        Square currentPath[MAX_N * MAX_N];
        Square bestPath[MAX_N * MAX_N];
        int currentLen = 0, bestLen = n * m;
        countWays = 0;
        bestPathSet = false;

        dfs(n, m, heights, 0, bestLen, currentPath, bestPath, currentLen);

        cout << "squares quantity " << bestLen << endl;
        for (int i = 0; i < bestLen; ++i) {
            cout << bestPath[i].x << " " << bestPath[i].y << " " <<
bestPath[i].w << endl;
        }
        cout << "ways: " << countWays << endl;

        return bestLen;
    }

private:
    int countWays;
    bool bestPathSet;

    void dfs(int n, int m, int heights[], int count, int& bestCount,
        Square currentPath[], Square bestPath[], int& currentLen) {

        if (count > bestCount) return;

        int left = 0, minHeight = m;
        for (int i = 0; i < n; i++) {
            if (heights[i] < minHeight) {
                minHeight = heights[i];
                left = i;
            }
        }

        if (minHeight == m) {
            if (count < bestCount) {
                bestCount = count;
                countWays = 1;
                for (int i = 0; i < count; ++i) {
                    bestPath[i] = currentPath[i];
                }
                bestPathSet = true;
            }
            else if (count == bestCount) {
                countWays++;
                if (!bestPathSet) {
                    for (int i = 0; i < count; ++i) {
                        bestPath[i] = currentPath[i];
                    }
                    bestPathSet = true;
                }
            }
            return;
        }

        int right = left;
        while (right < n && heights[right] == minHeight && (right - left +
minHeight) < m) {
            ++right;
```

```
        }

        int maxSize = right - left;
        if (m - minHeight < maxSize) maxSize = m - minHeight;

        for (int size = maxSize; size >= 1; size--) {
            if (count + 1 > bestCount) continue;
            if (size >= n || size >= m) continue;

            // Промежуточный вывод:
            cout << string(count * 2, ' ') << "[DFS level " << count << "]
Пытаемся поставить квадрат "
                << size << "x" << size << " в точку (" << left + 1 << ", " <<
minHeight + 1 << ")" << endl;

            for (int i = left; i < left + size; ++i)
                heights[i] += size;

            // Вывод текущего состояния heights[]
            cout << string(count * 2, ' ') << "Высоты после установки: [";
            for (int i = 0; i < n; ++i) cout << heights[i] << (i < n - 1 ? ",
" : "");
            cout << "]" << endl;

            currentPath[currentLen++] = { left + 1, minHeight + 1, size };

            dfs(n, m, heights, count + 1, bestCount, currentPath, bestPath,
currentLen);

            currentLen--;
            for (int i = left; i < left + size; ++i)
                heights[i] -= size;

            // Промежуточный вывод об откате:
            cout << string(count * 2, ' ') << "Откат после квадрата " << size
<< "x" << size
                << " в точке (" << left + 1 << ", " << minHeight + 1 << ")"
<< endl;
        }
    }

};

int main() {
    setlocale(LC_ALL, "Russian");
    Solution sol;
    int n, m;
    cin >> n >> m;
    sol.tilingRectangle(n, m);
    return 0;
}*/

int countWays = 0;
bool bestPathSet = false;

void dfs(int n, int m, int heights[], int cnt, int& bestCnt,
    Square currentPath[], Square bestPath[], int& currentLen) {

    if (cnt > bestCnt) return;

    // Поиск самой низкой колонки
    int left = 0, minHeight = m;
    for (int i = 0; i < n; i++) {
        if (heights[i] < minHeight) {
```

```
            minHeight = heights[i];
            left = i;
        }
    }

    // Если всё покрыто
    if (minHeight == m) {
        if (cnt < bestCnt) {
            bestCnt = cnt;
            countWays = 1;
            for (int i = 0; i < cnt; ++i) {
                bestPath[i] = currentPath[i];
            }
            bestPathSet = true;
        }
        else if (cnt == bestCnt) {
            countWays++;
            if (!bestPathSet) {
                for (int i = 0; i < cnt; ++i) {
                    bestPath[i] = currentPath[i];
                }
                bestPathSet = true;
            }
        }
        return;
    }

    // Поиск ширины свободной зоны справа
    int right = left;
    while (right < n && heights[right] == minHeight && (right - left +
minHeight) < m) {
        ++right;
    }

    int maxSize = right - left;
    if (m - minHeight < maxSize) maxSize = m - minHeight;

    for (int size = maxSize; size >= 1; size--) {
        if (cnt + 1 > bestCnt) continue;
        if (size >= n || size >= m) continue;


       /* cout << string(cnt * 2, ' ') << "[DFS level " << cnt << "] Пытаемся
поставить квадрат "
            << size << "x" << size << " в точку (" << left + 1 << ", " <<
minHeight + 1 << ")" << endl;*/

        for (int i = left; i < left + size; ++i)
            heights[i] += size;


    /*    cout << string(cnt * 2, ' ') << "Высоты после установки: [";
        for (int i = 0; i < n; ++i) cout << heights[i] << (i < n - 1 ? ", " :
"");
        cout << "]" << endl;*/

        currentPath[currentLen++] = { left + 1, minHeight + 1, size };
        dfs(n, m, heights, cnt + 1, bestCnt, currentPath, bestPath,
currentLen);
        currentLen--;

        for (int i = left; i < left + size; ++i)
            heights[i] -= size;
```

```
     /*   cout << string(cnt * 2, ' ') << "Откат после квадрата " << size <<
"x" << size
            << " в точке (" << left + 1 << ", " << minHeight + 1 << ")" <<
endl;*/
    }
}


int tilingRectangle(int n, int m) {
    if (n > m) swap(n, m);

    int heights[MAX_N] = { 0 };
    Square currentPath[MAX_N * MAX_N];
    Square bestPath[MAX_N * MAX_N];
    int currentLen = 0, bestLen = n * m;

    countWays = 0;
    bestPathSet = false;

    dfs(n, m, heights, 0, bestLen, currentPath, bestPath, currentLen);

    cout << "Минимальное количество квадратов: " << bestLen << endl;
    for (int i = 0; i < bestLen; ++i) {
        cout << bestPath[i].x << " " << bestPath[i].y << " " << bestPath[i].w
<< endl;
    }
    cout << "Количество способов заполнения прямоугольника минимльным
количеством квадратов: " << countWays << endl;

    return bestLen;
}

int main() {
    setlocale(LC_ALL, "Russian");
    int n, m;
    cin >> n >> m;
    tilingRectangle(n, m);
    return 0;
}
```