



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего
образования

«МИРЭА - Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий
Кафедра математического обеспечения и стандартизации
информационных технологий

ОТЧЕТ

ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 7

**«Алгоритмические стратегии или методы разработки алгоритмов. Перебор и
методы его сокращения.»**

по дисциплине

«Структуры и алгоритмы обработки данных»

Выполнил студент группы *ИКБО-03-22*

Хохлинов Д.И.

Принял

Сорокин А.В.

Практическая
работа выполнена

«__»_____2023 г.

«Зачтено»

«__»_____2023 г.

Москва 2023

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ	3
2 РАЗРАБОТКА ПРИЛОЖЕНИЯ	4
2.1 Метод «грубой силы» и оценка количества переборов	4
2.2 Метод динамического программирования	6
2.3 Класс field	8
2.4 Код программы.....	9
2.5 Тестирование	13
3 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	15

1 ПОСТАНОВКА ЗАДАЧИ

1. Разработать алгоритм решения задачи с применением метода, указанного в варианте и реализовать программу.

2. Оформить отчет, включив в него:

2.1. Условие задачи варианта и требования к выполнению задания.

2.2. Описание метода, предлагаемого к реализации в варианте.

2.3. Оценку количества переборов при решении задачи стратегией «в лоб» - грубой силы.

2.4. Привести анализ снижения числа переборов при применении метода

2.5. Отчет по разработке программы

2.6. Постановка задачи.

2.7. Тестовый пример решения.

2.8. Описание подхода к решению.

2.9. Функционал решения.

2.10. Код программы.

2.11. Результаты тестирования.

Персональный вариант:

Задача - Дано прямоугольное поле размером $n*m$ клеток. Можно совершать шаги длиной в одну клетку вправо или вниз. Посчитать, сколькими способами можно попасть из левой верхней клетки в правую нижнюю.

Метод – динамическое программирование.

2 РАЗРАБОТКА ПРИЛОЖЕНИЯ

2.1 Метод «грубой силы» и оценка количества переборов

В этом отчете метод решения задачи «в лоб» и метод динамического программирования будут разбираться для случая $n=3$, $m=3$ (получим квадратное поле 3×3). Тестирование будет выполняться для этого случая и для случаев 6×6 , 12×8 и 13×13 .

Попробуем решить данную задачу «в лоб». Рассмотрим поле, в котором нужно найти количество маршрутов между верхней левой и нижней правой клеткой (рисунок 1). Левая верхняя клетка отмечена красной звездочкой, правая нижняя – синей.

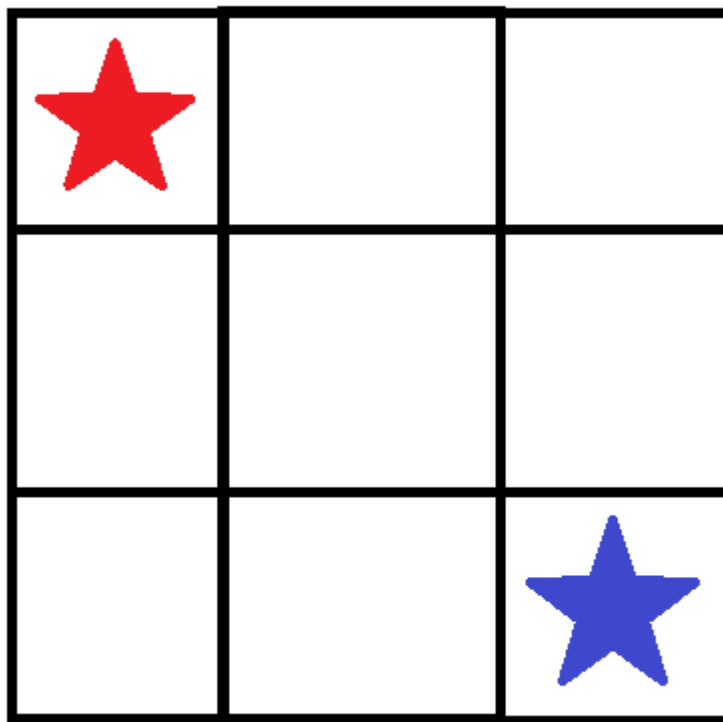


Рисунок 1 – Исходное поле

Очевидно, что для того, чтобы добраться от красной звездочки к синей, понадобится 2 движения вправо и 2 движения вниз. Обозначим движения вправо как нули (0), а движения вниз как единицы (1). Начальный маршрут (от которого будут получены все остальные) имеет вид 0011 и

изображен на рисунке 2, где движения вправо отмечены зелеными стрелками, а движения вниз – оранжевыми.

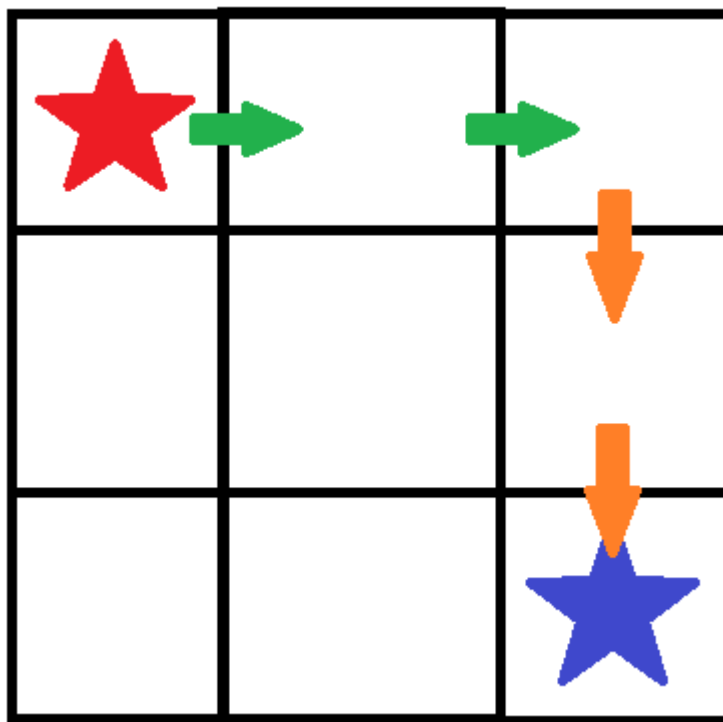


Рисунок 2 – Поле из рисунка 1 с отмеченным начальным маршрутом

Теперь необходимо каждый раз увеличивать «двоичное» значение маршрута на 1, пока не будет получено переполнение (появление в этом случае пятого разряда). После каждого увеличения необходимо проверить, равняется ли количество единиц в полученном двоичном числе количеству движений вниз (единиц) в начальном маршруте; если да, то нужно увеличить количество найденных путей на 1.

Применим описанный алгоритм к описанному случаю:

- 1) начальный случай: 0011, число путей - 1;
- 2) 0100 – не подходит;
- 3) 0101 – подходит, число путей – 2;
- 4) 0110 – подходит, число путей – 3;

5) 0111 – не подходит;

...

10) 1100 – подходит, число путей – 6;

11) 1101 – не подходит;

12) 1110 – не подходит;

13) 1111 – не подходит;

14) 10000 – переполнение – заканчиваем поиск.

Итоговое число путей – 6.

Оценка количества переборов методом «грубой силы»:

$$O(2^{n+m-2} - 2^{m-1}) = O(2^{n+m-2}) = O(2^{n+m}) \quad (1)$$

2.2 Метод динамического программирования

Рассмотрим другой метод решения описанной выше задачи. Суть метода состоит в разбиении задачи (нахождение количества путей для всего поля) на более простые подзадачи (нахождение количества путей для отдельно взятой ячейки), что существенно уменьшает время выполнения программы.

Рассмотрим этот метод для поля 3x3 (рисунок 3). Для левой верхней ячейки очевидно, что количество путей из левой верхней ячейки в эту ячейку равно 1. Для краткости будем обозначать ячейки как (x,y), где x – номер строки, y – номер столбца.

1		

Рисунок 3 – Исходное поле для метода динамического программирования

Затем будем проходить каждую ячейку слева направо и сверху вниз и определять для нее количество путей как сумму количества путей до ячейки слева и количества путей до ячейки сверху (если такие ячейки имеются). Получаем, что для ячеек (1,2), (1,3), (2,1) и (3,1) количество путей равно 1. Далее, для ячейки (2,2) количество путей равно сумме количества путей до ячейки (1,2) и до ячейки (2,1) (то есть $1+1=2$). Аналогично для ячейки (2,3) количество путей равно 3, для ячейки (3,2) – 3 и для ячейки (3,3) – 6 (рисунок 4).

1	1	1
1	2	3
1	3	6

Рисунок 4 – Поле после обхода всех ячеек

После заполнения всех ячеек значениями остается только получить значение в правой нижней ячейке. Ответ: 6 путей.

В результате был получен тот же ответ, что и с помощью метода «грубой силы», следовательно, можно говорить об эквивалентности этих методов.

Оценка сложности метода динамического программирования:

$$O(n * m) \quad (2)$$

2.3 Класс field

Для программного решения этой задачи определим класс field (листинг 1).

Листинг 1 – Класс field

```
class field {
    int size_x, size_y;
    int64_t** matrix;
public:
    field(int m, int n);
    ~field();
    void show();
    int64_t getNumberOfPaths();
    int64_t getNumberOfPathsBrute();
};
```

Пользователю доступны следующие методы:

1. Конструктор с параметрами: создает нулевую матрицу размера $m \times n$;
2. Деструктор: очищает память, выделенную под матрицу `matrix`;
3. Метод `show`: выводит матрицу на экран;
4. Метод `getNumberOfPaths`: возвращает количество путей из левой верхней ячейки в правую нижнюю, используя метод динамического программирования;
5. Метод `getNumberOfPathsBrute`: возвращает количество путей из левой верхней ячейки в правую нижнюю, используя метод «грубой силы».

2.4 Код программы

Приведем исходный код программы на C++ (листинг 2):

Листинг 2 – main.cpp

```
#include <iostream>
#include <locale.h>
#include <iomanip>
#include <chrono>

using namespace std;

class field {
    int size_x, size_y;
    int64_t** matrix;
public:
```

Продолжение листинга 2

```
    field(int m, int n);
    ~field();
    void show();
    int64_t getNumberOfPaths();
    int64_t getNumberOfPathsBrute();
};

field::field(int m, int n)
{
    this->size_x = m;
    this->size_y = n;
    this->matrix = new int64_t*[m];
    for (int i = 0; i < m; i++)
    {
        this->matrix[i] = new int64_t[n];
        for (int j = 0; j < n; j++)
        {
            this->matrix[i][j] = 0;
        }
    }
}

field::~~field() {
    for (int i = 0; i < size_x; i++)
    {
        delete[] matrix[i];
    }
    delete[] matrix;
}

int permutations_size_x = 0; //сколько нужно ходов вниз

bool permutate(int* moves, int size)
{
    bool matching = false, cango = true;
    //идем к следующему варианту
    while ((!matching) && (cango))
    {
        int i = size - 1;
        while (i >= 0)
        {
            if (moves[i] == 0)
            {
                moves[i] = 1;
                break;
            }
            else
            {
                moves[i] = 0;
                i--;
                if (i == -1)
                {
                    cango = false;
                    break;
                }
            }
        }
    }
}
```

Продолжение листинга 2

```
        }
    }
    //считаем единицы и ноли
    int onecount = 0, zerocount = 0;
    for (i = 0; i < size; i++)
    {
        (moves[i] == 1) ? onecount++ : zerocount++;
    }
    if (onecount == permutations_size_x)
    {
        matching = true;
    }
}
return matching;
}

int64_t field::getNumberOfPathsBrute() {
    if ((size_x == 1) || (size_y == 1))
        return 1;
    int* moves = new int[size_x + size_y - 2]; //0 - вправо, 1 - вниз
    for (int i = 0; i < size_y - 1; i++)
    {
        moves[i] = 0;
    }
    for (int i = size_y - 1; i < size_x + size_y - 2; i++)
    {
        moves[i] = 1;
    }
    int64_t count = 1;
    permutations_size_x = size_x - 1; //задаем необходимое количество
ходов вниз для метода
    while (permute(moves, size_x + size_y - 2))
    {
        count++;
    }
    return count;
}

int64_t field::getNumberOfPaths() {
    for (int i = 0; i < size_x; i++)
    {
        for (int j = 0; j < size_y; j++)
        {
            if ((i == 0) || (j == 0))
                matrix[i][j] = 1;
            else
                matrix[i][j] = matrix[i - 1][j] + matrix[i][j -
1];
        }
    }
    return matrix[size_x - 1][size_y - 1];
}

void field::show() {
    for (int i = 0; i < size_x; i++)
```

Продолжение листинга 2

```
{
    for (int j = 0; j < size_y; j++)
    {
        cout << setw(6) << matrix[i][j] << " | ";
    }
    cout << endl;
}

int main()
{
    setlocale(LC_ALL, "ru");
    int x = 14, y = 12; //высота и ширина
    field f = field(x, y);
    int64_t nopD = 0, nopB = 0;
    int timeD = 0, timeB = 0;

    auto start = chrono::steady_clock::now();
    nopD = f.getNumberOfPaths();
    auto end = chrono::steady_clock::now();

    timeD = chrono::duration_cast<chrono::nanoseconds> (end -
start).count();

    start = chrono::steady_clock::now();
    nopB = f.getNumberOfPathsBrute();
    end = chrono::steady_clock::now();

    timeB = chrono::duration_cast<chrono::microseconds> (end -
start).count();

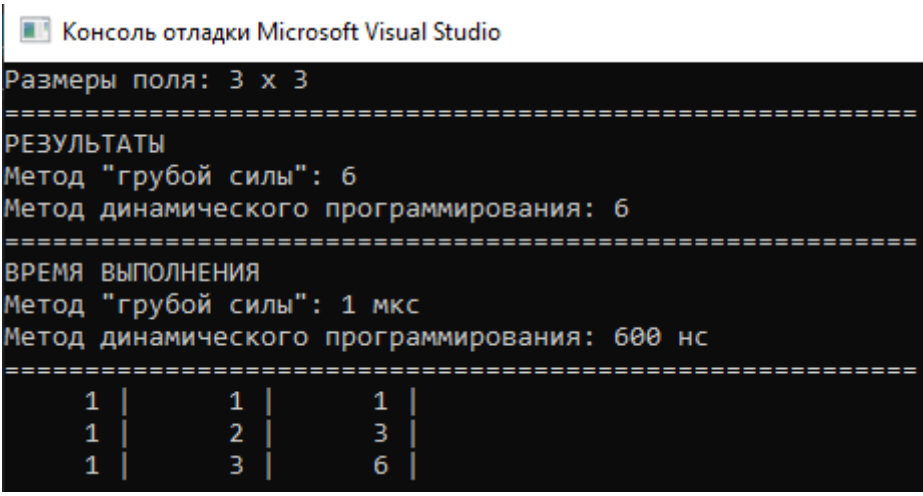
    cout << "Размеры поля: " << y << " x " << x << endl;
    cout <<
    "===== " << endl;
    cout << "РЕЗУЛЬТАТЫ" << endl;
    cout << "Метод \"грубой силы\": " << nopB << endl;
    cout << "Метод динамического программирования: " << nopD << endl;
    cout <<
    "===== " << endl;
    cout << "ВРЕМЯ ВЫПОЛНЕНИЯ" << endl;
    if (timeB > 1000000)
    {
        cout << "Метод \"грубой силы\": " << timeB / 1000000 << " с
" << (timeB % 1000000) / 1000 << " мс" << endl;
    }
    else if (timeB > 1000)
    {
        cout << "Метод \"грубой силы\": " << timeB / 1000 << " мс"
<< endl;
    }
    else {
        cout << "Метод \"грубой силы\": " << timeB << " мкс" <<
endl;
    }
}
```

Продолжение листинга 2

```
    cout << "Метод динамического программирования: " << timeD << "
нс" << endl;
    cout <<
"===== " << endl;
    f.show();
    return 0;
}
```

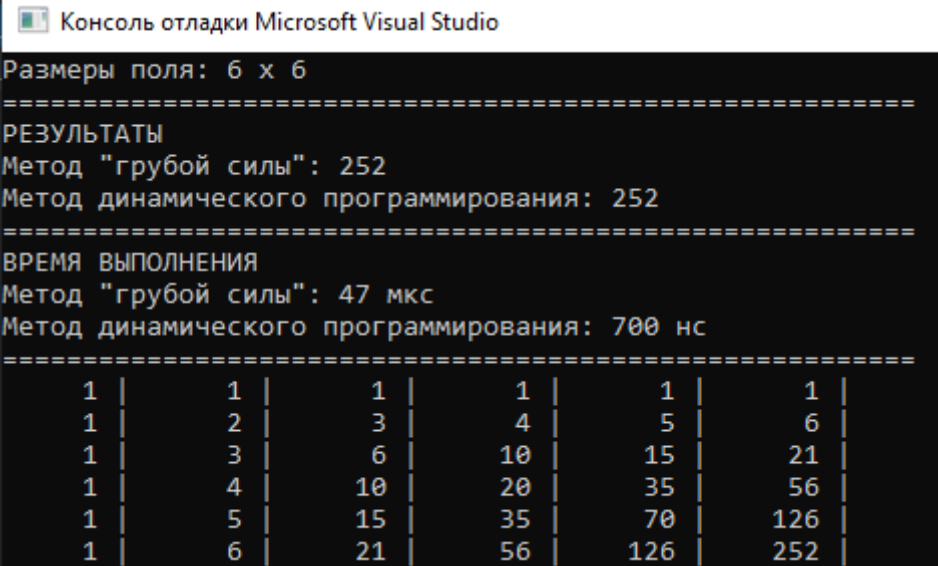
2.5 Тестирование

Проведем тестирование написанной программы для случаев 3x3, 6x6, 12x8 и 13x13 (рисунки 5-8):



```
Консоль отладки Microsoft Visual Studio
Размеры поля: 3 x 3
=====
РЕЗУЛЬТАТЫ
Метод "грубой силы": 6
Метод динамического программирования: 6
=====
ВРЕМЯ ВЫПОЛНЕНИЯ
Метод "грубой силы": 1 мкс
Метод динамического программирования: 600 нс
=====
1 | 1 | 1 |
1 | 2 | 3 |
1 | 3 | 6 |
```

Рисунок 5 – Результат для случая 3x3



```
Консоль отладки Microsoft Visual Studio
Размеры поля: 6 x 6
=====
РЕЗУЛЬТАТЫ
Метод "грубой силы": 252
Метод динамического программирования: 252
=====
ВРЕМЯ ВЫПОЛНЕНИЯ
Метод "грубой силы": 47 мкс
Метод динамического программирования: 700 нс
=====
1 | 1 | 1 | 1 | 1 | 1 |
1 | 2 | 3 | 4 | 5 | 6 |
1 | 3 | 6 | 10 | 15 | 21 |
1 | 4 | 10 | 20 | 35 | 56 |
1 | 5 | 15 | 35 | 70 | 126 |
1 | 6 | 21 | 56 | 126 | 252 |
```

Рисунок 6 - Результат для случая 6x6

```

Консоль отладки Microsoft Visual Studio

Размеры поля: 8 x 12
=====
РЕЗУЛЬТАТЫ
Метод "грубой силы": 31824
Метод динамического программирования: 31824
=====
ВРЕМЯ ВЫПОЛНЕНИЯ
Метод "грубой силы": 13 мс
Метод динамического программирования: 1100 нс
=====

```

1	1	1	1	1	1	1	1	1
1	2	3	4	5	6	7	8	
1	3	6	10	15	21	28	36	
1	4	10	20	35	56	84	120	
1	5	15	35	70	126	210	330	
1	6	21	56	126	252	462	792	
1	7	28	84	210	462	924	1716	
1	8	36	120	330	792	1716	3432	
1	9	45	165	495	1287	3003	6435	
1	10	55	220	715	2002	5005	11440	
1	11	66	286	1001	3003	8008	19448	
1	12	78	364	1365	4368	12376	31824	

Рисунок 7 – Результат для случая 12x8

```

Консоль отладки Microsoft Visual Studio

Размеры поля: 13 x 13
=====
РЕЗУЛЬТАТЫ
Метод "грубой силы": 2704156
Метод динамического программирования: 2704156
=====
ВРЕМЯ ВЫПОЛНЕНИЯ
Метод "грубой силы": 1 с 25 мс
Метод динамического программирования: 900 нс
=====

```

1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	2	3	4	5	6	7	8	9	10	11	12	13	
1	3	6	10	15	21	28	36	45	55	66	78	91	
1	4	10	20	35	56	84	120	165	220	286	364	455	
1	5	15	35	70	126	210	330	495	715	1001	1365	1820	
1	6	21	56	126	252	462	792	1287	2002	3003	4368	6188	
1	7	28	84	210	462	924	1716	3003	5005	8008	12376	18564	
1	8	36	120	330	792	1716	3432	6435	11440	19448	31824	50388	
1	9	45	165	495	1287	3003	6435	12870	24310	43758	75582	125970	
1	10	55	220	715	2002	5005	11440	24310	48620	92378	167960	293930	
1	11	66	286	1001	3003	8008	19448	43758	92378	184756	352716	646646	
1	12	78	364	1365	4368	12376	31824	75582	167960	352716	705432	1352078	
1	13	91	455	1820	6188	18564	50388	125970	293930	646646	1352078	2704156	

Рисунок 8 – Результат для случая 13x13

Из рисунков 7 и 8 видно, что увеличение ширины и высоты поля суммарно на 6 единиц увеличило время выполнения метода «грубой силы» в $\frac{1025}{13} \approx 78,846$ раз, и при этом $64 = 2^6 < 78,846 < 2^7 = 128$. Следовательно, оценка сложности метода грубой силы была верна.

3 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Задание на самостоятельную работу: https://online-edu.mirea.ru/pluginfile.php?file=%2F1144127%2Fmod_assign%2Fintroattachment%2F0%2FСиАОД%20Самостоятельная%20работа%207%20%28алгоритмические%20стратегии%29.pdf, дата обращения: 06.12.23
2. Структуры и алгоритмы обработки данных – лекция 2.8: https://online-edu.mirea.ru/pluginfile.php?file=%2F1126225%2Fmod_folder%2Fcontent%2F0%2F%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D1%8B%20%D0%B8%20%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B%20%D0%BE%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85%D0%9B%D0%B5%D0%BA%D1%86%D0%B8%D1%8F_2.8.pptx, дата обращения: 06.12.23