МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«МИРЭА – Российский технологический университет»**

**РТУ МИРЭА**

---

Отчет по выполнению практического задания №8.2

# Тема: Реализация алгоритмов на основе сокращения числа переборов

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент Зуев Д.А.
Группа ИКБО-68-23

**Москва 2025**

**Цель работы:** освоить приёмы работы с алгоритмами оптимизации

## ЗАДАНИЕ

### Формулировка задачи

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

2. Оценить количество переборов при решении задачи стратегией «в лоб» - грубой силы. Сравнить с числом переборов при применении метода.

Персональный вариант:

| № | Задача | Метод |
|---|---|---|
| 9 | Треугольник имеет вид, представленный на рисунке. Напишите программу, которая вычисляет наибольшую сумму чисел, расположенных на пути от верхней точки треугольника до его основания. | Динамическое программирование |

### Описание алгоритмов

### Метод динамического программирования

Динамическое программирование в теории управления и теории вычислительных систем — способ решения сложных задач путём разбиения их на более простые подзадачи.

Метод грубой силы

Метод грубой силы предполагает перебор всех возможных вариантов для нахождения решения. В данном случае, это может означать перебор всех клеток в пределах фигуры для вычисления её площади:

## Код программы

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int iterations = 0; // Счетчик итераций

// Рекурсивная функция
int maxPathSumRecursive(const vector<vector<int>>& triangle, int row = 0, int col = 0) {
    iterations++;


    if (row == triangle.size() - 1) {
        return triangle[row][col];
    }

    // Рекурсивно считаем два пути
    int leftPath = maxPathSumRecursive(triangle, row + 1, col);
    int rightPath = maxPathSumRecursive(triangle, row + 1, col + 1);


    return triangle[row][col] + max(leftPath, rightPath);
}
```

Рисунок 1 – реализация полного перебора

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int maxPathSum(vector<vector<int>>& triangle, int& iterations) {
    iterations = 0; // Обнуляем счетчик итераций

    // Начинаем с предпоследнего ряда и идем вверх
    for (int i = triangle.size() - 2; i >= 0; --i) {
        for (int j = 0; j < triangle[i].size(); ++j) {
            iterations++;
            triangle[i][j] += max(triangle[i + 1][j], triangle[i + 1][j + 1]);
        }
    }

    // Верхний элемент содержит максимальную сумму
    return triangle[0][0];
}
```

Рисунок 2 – реализация динамической программы

**Результаты тестирования**

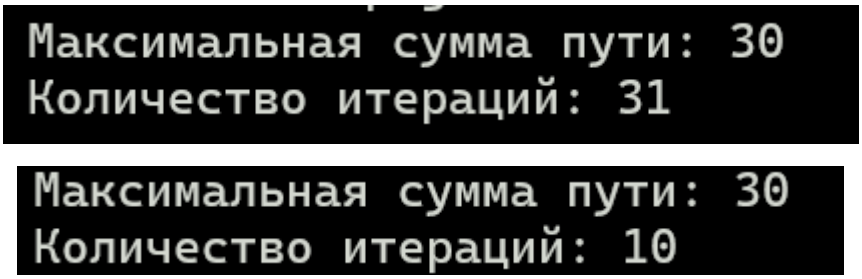


Рисунок 3 – вывод программы

**Оценка количества переборов**

**Метод грубой силы**

При использовании метода грубой силы количество переборов будет равно N, где N = $2^n$– 1. n - количество рядов.

**Метод динамического программирования**

Метод динамического программирования решает задачу более эффективно за время $n^2$

# ВЫВОД

Метод грубой силы основан на простом переборе всех элементов, что делает его интуитивно понятным, но неэффективным при больших размерах из-за значительного количества переборов. В отличие от него, метод динамического программирования позволяет значительно сократить количество проверок. Это делает его более оптимальным для данной задачи.

Подсчет количества переборов для обоих методов показал, что метод динамического программирования требует гораздо меньше операций, что подтверждает его эффективность. В результате тестирования было установлено, что данный подход не только ускоряет выполнение программы, но и делает её более масштабируемой.

Таким образом, работа продемонстрировала важность выбора алгоритмического подхода в зависимости от условий задачи. Метод динамического программирования оказался более предпочтительным.