Лабораторная работа № 7 по курсу дискретного анализа: «Динамическое программирование»

Выполнил студент группы М8О-307Б-21 МАИ Дубровин Дмитрий

Условие:

Вариант: 4 -> Игра с числом

Имеется натуральное число *n*. За один ход с ним можно произвести следующие действия:

- Вычесть единицу
- Разделить на два
- Разделить на три

При этом стоимость каждой операции – текущее значение *n*. Стоимость преобразования - суммарная стоимость всех операций в преобразовании. Вам необходимо с помощью последовательностей указанных операций преобразовать число *n* в единицу таким образом, чтобы стоимость преобразования была наименьшей. Делить можно только нацело.

Формат ввода

В первой строке строке задано 2 <= n <= 10^7

Формат вывода

Выведите на первой строке искомую наименьшую стоимость. Во второй строке должна содержаться последовательность операций. Если было произведено деление на 2 или на 3, выведите /2 (или /3). Если же было вычитание, выведите -1. Все операции выводите разделяя пробелом.

Пример

Ввод	**Вывод**
82	202 -1 /3 /3 /3 /3

Метод решения

Для решения поставленной задачи воспользуемся методом динамического программирования.

Динамическое программирование (DP) - это метод решения задач, который использует подход "разделяй и властвуй". Основная идея DP заключается в том, что сложная задача разбивается на более простые подзадачи, которые решаются

независимо, и затем результаты комбинируются для нахождения ответа на исходную задачу.

1. В данной задаче нахождения минимальной стоимости преобразования числа n в 1 с использованием операций вычитания 1, деления на 2 и деления на 3, мы можем применить DP следующим образом:

2. Создаем массив cost, где cost[i] будет хранить минимальную стоимость преобразования числа i в 1. Начальное значение стоимости для 1 равно 0 (так как для числа 1 не нужны операции).

3. Мы итерируемся от 2 до n и для каждого числа i рассматриваем три возможных операции: вычитание 1, деление на 2 и деление на 3.

4. Для каждой из этих операций мы вычисляем стоимость, которая равна текущей стоимости числа i (cost[i]) плюс стоимости предыдущего числа (cost[i - 1] для вычитания, cost[i / 2] для деления на 2 и cost[i / 3] для деления на 3). Мы выбираем операцию с минимальной стоимостью и обновляем cost[i] этой минимальной стоимостью.

5. Таким образом, после завершения итераций от 2 до n, cost[n] будет содержать минимальную стоимость преобразования числа n в 1.

6. Затем мы можем восстановить последовательность операций, начиная с числа n и двигаясь назад к 1. Для этого мы используем массив prevAction, который хранит информацию о предыдущей операции для каждого числа.

7. Фактически, мы начинаем с числа n, идем назад к 1, и для каждого числа определяем, какая операция была выбрана на предыдущем шаге (вычитание, деление на 2 или деление на 3) и добавляем ее к последовательности операций.

8. В итоге мы получаем последовательность операций, которая минимизирует стоимость преобразования числа n в 1.

Тест производительности

Сложность данной программы можно разбить на несколько аспектов:

1. Заполнение массивов cost и prevAction происходит в цикле for, который выполняется n-1 раз (от 2 до n), где n - входное число. Это дает сложность O(n).

2. Восстановление последовательности действий также происходит в цикле, который выполняется не более n раз (поскольку мы начинаем с n и двигаемся к 1). Это также дает сложность O(n).

3. Вывод последовательности действий также занимает O(n) операций, так как мы выводим последовательность действий в обратном порядке.

Таким образом, общая сложность программы составляет O(n), где n - входное число. В данном случае, это линейная сложность относительно n, что означает, что время выполнения программы будет линейно зависеть от размера входных данных.

Для проверки времени ее работы я написал несколько входных тестов, а также немного изменил код, для считывание информации из файла. Запись времени работы производится в отдельный файл output.txt.

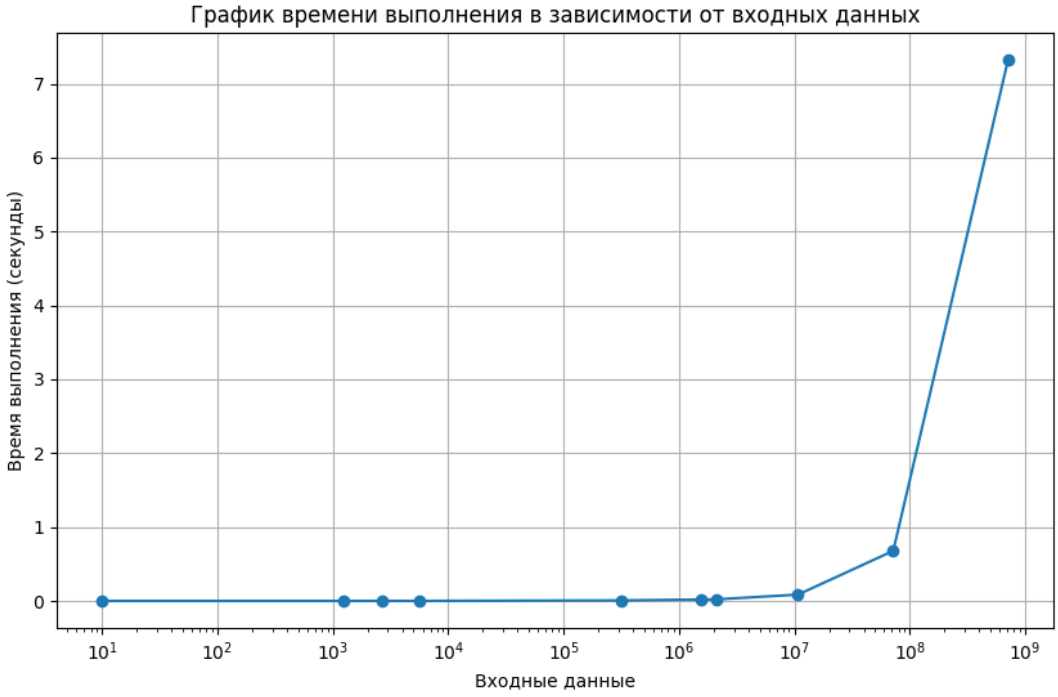

Рисунок 1 - График времени работы программы относительно входных данных

Как видно из график время работы сильно возрастает при наличии входных данный больше 10 ^ 7, что противоречит условию задачи, так как на вход нам подаются числа меньше 10 ^ 7.

Вывод

Выполнив лабораторную работу №7 по курсу «Дискретный анализ», я изучил динамическое программирование и вспомнил C++. Я убедился, что динамическое программирования помогает в оптимизации решения задачи, благодаря сохранению предыдущих результатов, что позволяет не пересчитывать одни и те же случаи. Однако для некоторых задач ДП может требовать слишком много памяти, что приведет к переполнению. Из-за чего следует правильно подходить к решению задач.