Лабораторная работа №7 по курсу дискретного анализа: Игра с числом

Выполнил студент МАИ группы М8О-201Б *Ефимов Александр.*

Постановка задачи

При помощи метода динамического программирования разработать алгоритм решения задачи, определяемой своим вариантом; оценить время выполнения алгоритма и объем затрачиваемой оперативной памяти. Перед выполнением задания необходимо обосновать применимость метода динамического программирования.

Разработать программу на языке C или C++, реализующую построенный алгоритм. Формат входных и выходных данных описан в варианте задания:

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.

Метод решения

В каждый момент времени можно выполнить одно из трех действий, после выполнения которых стоимость операций увеличивается на текущее число. Т.к. необходимо найти наименьшую стоимость, то задачу можно свести к выполнению наименьшего числа операций. Это означает, что для каждого числа, над которым выполняется действие, необходимо выполнить такое действие, что новое число уже имеет наименьшую стоимость (т.е. к этому числу пришли за наименьшее количество действий). Значит оптимальное решение каждого числа можно найти рекурсивно.

Если же новое число не является оптимальным решением, значит можно найти другое число, которое решается за меньшее количество действий.

Если решать проблему рекурсивно для каждого полученного числа после каждого выполненного действия, то скорее всего придется находить оптимальное решение для одних и тех же чисел, например: от числа 18 можно дойти до числа 6 минимум двумя методами: $18 \div 2 - 1 - 1 - 1 = 6$ и $18 \div 3 = 6$. Т.к. проблема решается рекурсивно, то придется найти оптимальное решение для числа 6 дважды.

Во избежание этого, можно ввести дополнительную структуру, которая будет содержать решение для любого $i \in [2, n]$ (лучше всего, к которой можно обратиться за $O(1)$ время). Так как решение каждого числа i будет зависеть от одного из чисел до промежутка $[2, i]$, то для получения всех оптимальных решений достаточно найти все оптимальное решение каждого числа от 2 до n, при этом сохраняя информацию о том,

от какого числа получено это решение, после чего можно цепочкой построить полное решение задачи начиная с n.

Описание программы

Все решение содержится в одном файле *main.cpp*.

После принятия необходимого числа и объявления вектора типа *TOptimalNode* (для числа $i \in [2, n]$ содержит стоимость всех действий и наилучшее действие для этого числа), программа в цикле ищет лучшее решение каждого числа до n включительно: для числа i среди $i/2$, $i/3$ и $i - 1$ ищет то, которое имеет наименьшую стоимость, и записывает в вектор под индекс i действие, необходимое для получения лучшего числа (может принимать одно из четырех значений, заданным типом *TAction*), а также новую стоимость.

После подсчета всех значений, выводится стоимость числа n и в цикле выводится по цепочки все действия до числа 1.

Дневник отладки

Номер	Ошибка	Обнаруженная причина	Исправление
1	Зацикливание на -1	Во втором цикле третье выражение $(++i)$ увеличивало индекс после всех действий	Убрать третье выражение, оставляя только начальное действие цикла и условие его продолжения

Тест производительности

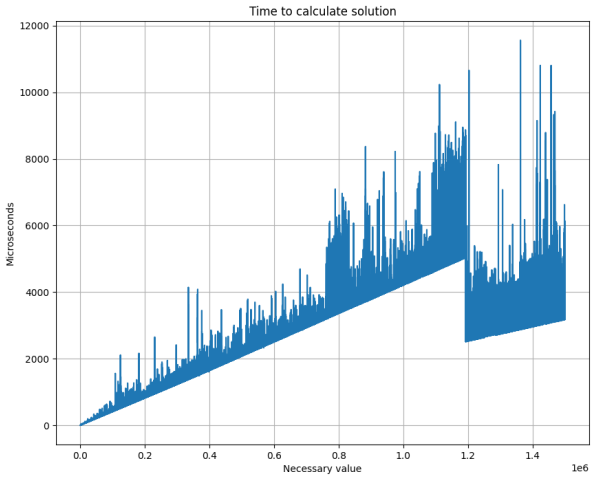

Рис. 1: Время решения.

На рисунке показано время решения в микросекундах для числа n (число указано на абсциссе). Сложность практически линейная, за исключением 1192762 и далее, которые, возможно из-за оптимизации компилятором, ускорились на 2500 микросекунд.

Выводы

Некоторые алгоритмы, такие как высчитывание биномиальных коэффициентов или поиск оптимальных (минимальных или максимальных), требуют рекурсивного вычисления меньших значений, но эти значения могут быть решены не один раз, что наделяет программу лишней сложностью. Тогда может появиться идея – просто записывать повторные значения в память. Такой подход получил названия динамического программирования:

Динамическое программирования является одним из подходов к решению, при котором задача разделяется на меньшие, а их решения записаны на тот случай, если при разделении у разных задач может появиться одна и та же проблема (причем должно быть доказано, что такая ситуация имеет место, иначе записанные решения просто будут тратить память. Это увеличивает пространственную сложность, но значительно ускоряет подсчеты (если доступ к нужным элементам достаточно быстрый).

Пример использования:

- Чаще всего, если нужно вычисление биномиальных коэффициентов, то оно нужно более одного раза. Поэтому проще решать задачу сверху вниз, чтобы решать только необходимые на данный момент коэффициенты, и записывать промежуточные решения на случай, если какое-либо из них понадобится снова.

- Дан набор монет, с помощью которых нужно набрать определенную сумму. Здесь нельзя набирать просто монеты наибольшего номинала, так как можно дойти до состояния, где итоговая сумма будет всегда больше. Например для набора $\{1, 3, 3, 5, 5\}$ нужно набрать сумму 12. Если начать брать наибольший номинал, то суммы могут быть $5 + 5 + 3 = 13$ и $5 + 5 + 1 = 11$. Если искать сумму как минимальная среди сумм, где вычтен один из доступных номиналов, то результат будет $5 + 3 + 3 = 12$, но одни и те же суммы можно высчитать несколько раз, поэтому их нужно запоминать.