Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №7 по курсу «Дискретный анализ»

Студент: Ю. Ю. Обыденкова
Преподаватель: А. Н. Ридли
Группа: М8О-308Б-18
Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №7

Задача: 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.
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 входных данных: В первой строке задано число $2 \leq n \leq 10^7$

1 Описание

Решим задачу методом динамического программирования. Заметим, что имеется оптимальная подструктура решения — его можно свести к решению подзадач меньшего размера. Действительно, минимальная стоимость $p[n]$ преобразования числа n равна $n + min(p[n/2], p[n/3], p[n-1])$. Стоимость $p[n/2]$ или $p[n/3]$ считается бесконечной, если n не делится на 2 или 3 соответственно, так как деление происходит только нацело. При построении рекурсивного решения становится ясно, что имеет место перекрытие вспомогательных подзадач, значит оптимальное решение можно построить методом восходящего анализа. Оптимальное решение заключается в нахождении решения для всех чисел от 1 до n последовательно, с сохранением результата для каждого числа в этом промежутке. Таким образом, решение имеет оценку $O(n)$ для времени и памяти.

2 Исходный код

После считывания числа, запускается цикл, заполняющий вектор с решениями для чисел от 2 до n. В этом же цикле заполняется вектор, предназначенный для восстановления ответа.

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

#define MAXINT 2147483647

enum PathElement {
    MinusOne, DivByTwo, DivByThree
};

std::string PathElementToString(PathElement elem) {
    if (elem == MinusOne) {
        return "-1";
    }
    if (elem == DivByTwo) {
        return "/2";
    }
    if (elem == DivByThree) {
        return "/3";
    }
    throw std::logic_error("Unknown element");
}

int GoToNextPathElement(int n, PathElement elem) {
    if (elem == MinusOne) {
        return n - 1;
    }
    if (elem == DivByTwo) {
        return n / 2;
    }
    if (elem == DivByThree) {
        return n / 3;
    }
    throw std::logic_error("Unknown element");
}

int main() {
    int n;
    std::cin >> n;
    std::vector<int> dp(n + 1);
    std::vector<PathElement> path(n + 1);
    dp[1] = 0;
```

```
    for (int i = 2; i <= n; ++i) {
        int min_one = dp[i - 1];
        int div_by_2 = (i % 2 == 0 ? dp[i / 2] : MAXINT);
        int div_by_3 = (i % 3 == 0 ? dp[i / 3] : MAXINT);

        dp[i] = std::min(min_one, std::min(div_by_2, div_by_3));
        if (dp[i] == min_one) {
            path[i] = MinusOne;
        } else if (dp[i] == div_by_2) {
            path[i] = DivByTwo;
        } else {
            path[i] = DivByThree;
        }
        dp[i] += i;
    }
    std::cout << dp[n] << "\n";
    int index = n;
    while (index != 1) {
        std::cout << PathElementToString(path[index]) << " ";
        index = GoToNextPathElement(index, path[index]);
    }
    std::cout << "\n";
    return 0;
}
```

3 Консоль

```
ulia@WIN-9LNCMF0CCPQ:~$ g++ lab7.cpp -o lab7
ulia@WIN-9LNCMF0CCPQ:~$ ./lab7
82
202
-1 /3 /3 /3 /3
ulia@WIN-9LNCMF0CCPQ:~$ ./lab7
500
1143
/2 /2 -1 /2 /2 -1 /3 /2 -1 /2 -1
ulia@WIN-9LNCMF0CCPQ:~$ ./lab7
3123523
7818750
-1 /3 /3 /3 /3 /3 /2 -1 /3 /3 /3 /2 -1 -1 /3 /3 -1 /3 /2 -1
```

4 Тест производительности

Замерим время выполенения программы для различных входных данных

```
ulia@WIN-9LNCMF0CCPQ:~$ ./lab7
82
202
-1 /3 /3 /3 /3
Time 1 ms
ulia@WIN-9LNCMF0CCPQ:~$ ./lab7
205
558
-1 /3 /2 /2 -1 /2 /2 /2 -1
Time 2 ms
ulia@WIN-9LNCMF0CCPQ:~$ ./lab7
50000
103703
/2 /2 /2 /2 -1 /2 /2 -1 /3 /2 /2 -1 /2 /2 /2 /2 /2 -1
Time 1 ms
ulia@WIN-9LNCMF0CCPQ:~$ ./lab7
1000000
2008788
/2 /2 /2 /2 /2 /2 -1 /3 /3 /2 /2 /2 -1 /3 /3 /3 /2 /2 -1
Time 20 ms
ulia@WIN-9LNCMF0CCPQ:~$ ./lab7
10000000
20079033
/2 /2 /2 /2 /2 /2 -1 /3 /3 /3 /3 /3 -1 /3 /2 -1 /2 -1 /2 /2 -1 /3 /2 -1
Time 180 ms
ulia@WIN-9LNCMF0CCPQ:~$
```

Очевидно, что оценка временной сложности программы была верна.

5 Выводы

Выполнив седьмую лабораторную работу по курсу «Дискретный анализ», я научилась решать задачи методом динамического программирования: искать оптимальную подструктуру и составлять рекурсивное решение, когда подпрограмма вызывает сама себя, и нерекурсивное решение.

Динамическое программирование стоит применять для решения задач, которые обладают двумя характеристиками:

1. Можно составить оптимальное решение задачи из оптимального решения ее подзадач.
2. Рекурсивный подход к решению проблемы предполагал бы многократное (не однократное) решение одной и той же подпроблемы, вместо того, чтобы производить в каждом рекурсивном цикле все новые и уникальные подпроблемы.

Так, в решаемой мной задаче методом динамического программирования, можно заметить, что имеется оптимальная подструктура решения — его можно свести к решению подзадач меньшего размера. Действительно, минимальная стоимость p[n] преобразования числа n равна n + min(p[n/2], p[n/3], p[n 1]). Стоимость p[n/2] или p[n/3] считается бесконечной, если n не делится на 2 или 3 соответственно, так как деление происходит только нацело. При построении рекурсивного решения становится ясно, что имеет место перекрытие вспомогательных подзадач, значит оптимальное решение можно построить методом восходящего анализа. Таким образом, решение имеет оценку O(n) для времени в памяти.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))