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: Н. О. Тимофеева
Преподаватель: И. Н. Симахин
Группа: М8О-308Б-19
Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №7

Задача: При помощи метода динамического программирования разработать алгоритм решения задачи, определяемой своим вариантом. Разработать программу на языке C или C++, реализующую построенный алгоритм.

Вариант: 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 строке задано $2 <= n <= 10^7$.

Формат результата

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.

1 Описание

Как описано в [1], динамическое программирование - это метод решения задач, при котором сложная задача разбивается на более простые, решение сложной задачи состоит из решений простых задач. Динамическое программирование допускает использование метода восходящего анализа, который позволяет изначально решать простые задачи и получать на их базе решение более сложных. Так же метод запоминает решения подзадач, потому что часто для построения приходится обращаться за оптимальным решением к одним и тем же малым задачам.

Динамическое программирование, как правило, применяется к задачам оптимизации. Такая задача может иметь много возможных решений. С каждым вариантом решения можно сопоставить какое-то значение, нам нужно найти среди них решение с оптимальным (минимальным или максимальным) значением.

Рассмотрим нашу задачу.

Будем решать её методом восходящего анализа. Для этого будем идти от 2 до n и находить стоимость преобразования, используя ранее полученные результаты. Таким образом мы сократим время выполнения работы по сравнения с наивным алгоритмом, так как будем идти по минимальным путям и не будем пересчитывать значения. Затем восстановим путь из n в 1, выбирая из чисел, полученных вычитанием 1-цы или делением на 2 или 3, то, у которого стоимость наименьшая.

2 Исходный код

Создаём вектор cost, в котором будет храниться стоимости преобразования чисел от 0, 1, ..., n-1, n к единице. В цикле for ищем эти стоимости, с помощью цикла while восстанавливам путь преобразования из n в единицу.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    int n;
    cin >> n;
    vector<long long> cost(n + 1, 0);
    for (int i = 2; i <= n; ++i) {
        if (i % 2 == 0 && i % 3 == 0) {
            cost[i] = i + min(cost[i - 1], min(cost[i / 2], cost[i / 3]));
        }
        else if (i % 2 == 0) {
            if (cost[i / 2] < cost[i - 1]) {
                cost[i] = i + cost[i / 2];
            }
            else {
                cost[i] = i + cost[i - 1];
            }
        }
        else if (i % 3 == 0) {
            if (cost[i / 3] < cost[i - 1]) {
                cost[i] = i + cost[i / 3];
            }
            else {
                cost[i] = i + cost[i - 1];
            }
        }
        else {
            cost[i] = i + cost[i - 1];
        }
    }
    cout << cost[n] << '\n';
    while (n > 1) {
        if (n % 2 != 0 && n % 3 != 0) {
            cout << "-1 ";
            n = n - 1;
        }
        else if (n % 3 != 0) {
            if (cost[n / 2] < cost[n - 1]) {
                cout << "/2 ";
                n = n / 2;
            }
            else {
                cout << "-1 ";
                n = n - 1;
            }
        }
        else if (n % 2 != 0) {
            if (cost[n / 3] < cost[n - 1]) {
                cout << "/3 ";
                n = n / 3;
            }
            else {
                cout << "-1 ";
```

```
57                 n = n - 1;
58             }
59         }
60         else {
61             if (cost[n / 3] < cost[n - 1] && cost[n / 3] < cost[n / 2]) {
62                 cout << "/3 ";
63                 n = n / 3;
64             }
65             else if (cost[n / 2] < cost[n - 1] && cost[n / 2] < cost[n / 3]) {
66                 cout << "/2 ";
67                 n = n / 2;
68             }
69             else {
70                 cout << "-1 ";
71                 n = n - 1;
72             }
73         }
74     }
75     cout << '\n';
76 }
```

3 Консоль

```
natalya@natalya-Ideapad-Z570:~/DA/Lab7$ ./a.out
2001
3721
/3 -1 /3 /3 /2 -1 /3 /3 /2 -1
natalya@natalya-Ideapad-Z570:~/DA/Lab7$ ./a.out
4620724
10066397
/2 /2 -1 /3 /2 /2 -1 /3 /3 /2 /2 -1 /3 /3 /3 /3 /3 -1 /2 -1 /2 -1
```

4 Тест производительности

Для измерения производительности подсчитываем время работы наивного алгоритма и метода динамического программирования. Замер времени работы производится с помощью библиотеки chrono. Проверим время работы на тестах с разными числами: 100, 1000, 10000, 100000, 1000000, 10000000.

```
natalya@natalya-Ideapad-Z570:~/DA/Lab7$ ./a.out <test.txt
100
DP: 52 ms
Naive: 273 ms

natalya@natalya-Ideapad-Z570:~/DA/Lab7$ ./a.out <test.txt
1000
DP: 74 ms
Naive: 643726 ms

natalya@natalya-Ideapad-Z570:~/DA/Lab7$ ./a.out <test.txt
10000
DP: 206 ms
Naive: 64582859 ms

natalya@natalya-Ideapad-Z570:~/DA/Lab7$ ./a.out <test.txt
100000
DP: 699 ms

natalya@natalya-Ideapad-Z570:~/DA/Lab7$ ./a.out <test.txt
1000000
DP: 8372 ms

natalya@natalya-Ideapad-Z570:~/DA/Lab7$ ./a.out <test.txt
10000000
DP: 68191 ms
```

Как можно заметить метод динамического программирования сильно выигрывает по времени у наивного алгоритма, что говорит о его высокой эффективности.

5 Выводы

Выполнив седьмую лабораторную работу по курсу «Дискретный анализ», я познакомилась с методом динамического программирования.

Имеется два способа реализации метода динамического программирования.

Первый способ - нисходящий с запоминанием. При таком подходе мы пишем процедуру рекурсивно, но так, чтобы она запоминала решение каждой подзадачи.

Второй способ - восходящий. Каждую подзадачу мы решаем только один раз, и к моменту, когда мы впервые с ней сталкиваемся, все необходимые для её решения подзадачи уже найдены.

Динамическое программирование позволяет разработать точные и относительно быстрые алгоритмы для решения сложных задач, в то время, как переборное решение слишком медленное.