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: С. А. Красоткин
Преподаватель: Н. С. Капралов
Группа: М8О-208Б
Дата:
Оценка:
Подпись:

Москва, 2021

0.0 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 и разработать программу на языке C++, реализующую построенный алгоритм.

Вариант задания: `4`

Описание: 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.

0.1 Описание

Я буду решать задачу методом восходящего динамического программирования. А именно решу подзадачи, а потом соберу из них решение общей.

В самом деле из числа a может быть получено по возможности три числа (обозначу сумму затраченную для i-го числа как $\widehat{s_i}$):

1. Если $a \equiv 0 \pmod 3$, то для числа $b = \frac{a}{3}$, $\widehat{s_b} = min(s_b, s_a + a)$. Но так как деление на три всегда опережает деление на два и вычитание единицы, то это операцию поиска минимума можно оптимизировать до $\widehat{s_b} = s_a + a$.
2. Если $a \equiv 0 \pmod 2$, то для числа $b = \frac{a}{2}$, $\widehat{s_b} = min(s_b, s_a + a)$.
3. Всегда для числа $b = a - 1$, $\widehat{s_b} = min(s_b, s_a + a)$.

Таким образом показано, что существует решение для чисел из набора $F = \{1, .., n - 1\}$.

Приступая к оценке времени, разобью общую задачу на две условных подзадачи *поиск ответа* и *поиск пути ответа*.

- Решается задача для каждого числа из набора F, значит нужно помнить все промежуточные состояния, а поэтому придётся выделить память для мемоизации $8 * 10000000$ байт, что 78.125 мегабайт (на практике получается в 4 раза меньше), то есть $O(n)$. Всего я решаю n подзадач, значит, и время работы программы $O(n)$
- Поиск пути ответа оптимизруется за счёт мемоизации. Общее число вызовов это: $a \cdot \frac{n}{3} + b \cdot \frac{n}{2} + c$, где a, b, c - число необходимых операций. Первые два коэффиента в среднем можно оценить $log_3(n)$ и $log_2(n)$, а значит общее число вызовов: $nln(n)\frac{ln108}{6ln2 \cdot ln3} + c$, таким образом $\Theta(nlnn)$.

0.2 Исходный код

Сначала выделю массив типа long long для мемоизации. Затем запощу цикл по n для поиска ответа, после чего буду его искать с помощью массива.

```
#include <iostream>
#include <algorithm>

using namespace std;

const int MAX_INT = 1e7+3;
long long dp[MAX_INT];

int Init(int n) {
    for(int i = 0; i <= n; i++) {
        dp[i] = 0;
    }

    return 0;
}

int main() {
    long long sum = 0;
    string answer = "";
    int n;
    cin>>n;
    Init(n);
    for(int i = n; i > 1; i--) {
        sum = dp[i] + i;
        if(i % 3 == 0) {
            dp[i / 3] = sum;
        }
        if(i % 2 == 0) {
            (dp[i / 2] == 0) ? dp[i / 2] = sum : dp[i / 2] = min(dp[i / 2], sum);
        }
        (dp[i - 1] == 0) ? dp[i - 1] = sum : dp[i - 1] = min(dp[i - 1], sum);
    }

    cout<<dp[1]<<"\n";

    int i = 1;
    while(i != n) {
        sum = dp[i];
        if( 3 * i <= n && (sum - i * 1ll * 3 ) == dp[3*i] ) {
            answer += " 3/";
            i *= 3;
        }
        else if( 2 * i <= n && (sum - i * 1ll * 2 ) == dp[2*i] ) {
            answer += " 2/";
            i *= 2;
```

```
            }
            else if( (sum - i - 1) == dp[i+1]) {
                answer += " 1-";
                i++;
            }
        }

        reverse(answer.begin(), answer.end());
        cout<<answer<<"\n";

        return 0;
}
```

0.3 Консоль

```
goku@debian: g++ sndAttempt.cpp
goku@debian: cat ../../../LR-s/7/Tests/3.txt
10000000
goku@debian: ./a.out < ../../../LR-s/7/Tests/3.txt
20079033
/2 /2 /2 /2 /2 /2 -1 /3 /3 /3 /3 /3 -1 /3 /2 -1 /2 -1 /2 /2 -1 /3 /2 /2
```

0.4 Тест производительности

Для сравнения написал переборный вариант нахождения пути от 1 к n.

```
goku@debian:~/Documents/Vuz/MAI/4sem/DA/Labs/7/Checkers$ g++ sndAttempt.cpp
goku@debian:~/Documents/Vuz/MAI/4sem/DA/LR-s/7/20210404$ g++ brute.cpp -o br
goku@debian:~/Documents/Vuz/MAI/4sem/DA/Labs/7/Checkers$ ./a.out < ../../../LR-s/7/Tests/2.txt
Find solution: 0
3
Find path: 0
/3
Time execution: 0
goku@debian:~/Documents/Vuz/MAI/4sem/DA/LR-s/7/20210404$ ./br < ../Tests/2.txt
Find solution: 0
3
Find path: 0
/3
Time execution: 0
goku@debian:~/Documents/Vuz/MAI/4sem/DA/Labs/7/Checkers$ ./a.out < ../../../LR-s/7/Tests/9.txt
Find solution: 2
203703
Find path: 0
/2 /2 /2 /2 /2 -1 /2 /2 -1 /3 /2 /2 -1 /2 /2 /2 /2 /2 /2
Time execution: 3
goku@debian:~/Documents/Vuz/MAI/4sem/DA/LR-s/7/20210404$ ./br < ../Tests/9.txt
n: 100000
Find solution: 3
203703
/2 /2 /2 /2 /2 -1 /2 /2 -1 /3 /2 /2 -1 /2 /2 /2 /2 /2 /2
Find path: 0
Time execution: 5
goku@debian:~/Documents/Vuz/MAI/4sem/DA/Labs/7/Checkers$ ./a.out < ../../../LR-s/7/Tests/10.txt
Find solution: 7
2008788
Find path: 0
/2 /2 /2 /2 /2 /2 -1 /3 /3 /2 /2 /2 -1 /3 /3 /3 /2 /2 /2
Time execution: 15
goku@debian:~/Documents/Vuz/MAI/4sem/DA/LR-s/7/20210404$ ./br < ../Tests/10.txt
n: 1000000
Find solution: 9
2008788
/2 /2 /2 /2 /2 /2 -1 /3 /3 /2 /2 /2 -1 /3 /3 /3 /2 /2 /2 Find path: 0
Time execution: 15
goku@debian:~/Documents/Vuz/MAI/4sem/DA/Labs/7/Checkers$ ./a.out < ../../../LR-s/7/Tests/11.txt
Find solution: 16
5081519
Find path: 0
-1 /3 /3 /3 /2 -1 /2 /2 -1 /3 /2 -1 /3 /2 -1 /2 /2 /2 /2 /2 /2 /2 /2
Time execution: 26
goku@debian:~/Documents/Vuz/MAI/4sem/DA/LR-s/7/20210404$ ./br < ../Tests/11.txt
```

```
n: 1999999
Find solution: 16
5081519
-1 /3 /3 /3 /2 -1 /2 /2 -1 /3 /2 -1 /3 /2 -1 /2 /2 /2 /2 /2 /2 /2 /2 Find path: 0
Time execution: 26
```

Как видно, переборная реализация поиска пути проигрывает динамической за счёт мемаизации. Проблема моей реализации в том, что я использую ревёрс строки для поиска ответа, что влияет на общее время выполнения программы.

0.5 Выводы

По выполнению 7-й лабораторной работы использовал библиотеку chrono для теста производительности. Чего-то сильно лучшего чем утилита time не почувствовал, но, возможно, это объясняется тем, что полезнее применять средства языка, как можно чаще.

Литература

[1] Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн *Алгоритмы Построение и анализ* — . Третье издание, Москва, Санкт-Петербург, Киев, 2013, 1324 с.