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: А. С. Бычков
Преподаватель: Н. К. Макаров
Группа: М8О-301Б

Дата:
Оценка:
Подпись:

Москва, 2024

Лабораторная работа №7

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

1. Вычесть единицу
2. Разделить на два
3. 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.

Форма 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.

1 Описание

Идея решения очень проста:

1. Если мы в 1, то нам ничего делать не нужно, можем вернуть ноль.
2. Иначе, текущее число больше единицы, значит его нужно уменьшать. Сделать это можно следующими способами:
   (a) Вычесть единицу из текущего числа.
   (b) Если текущее число четно - то можем разделить его на 2.
   (c) Если текущее число кратно 3 - можем разделить его на 3.

Итого, надо на каждом шаге алгоритма выбирать действие которым мы получим число, которое имеет наимешьную цену. К полученному минимуму надо прибавить текущее число.

Терминальным условием рекурсии будет то, что текущее число равно 1. В этом случае будем возвращать нолью

2 Исходный код

```cpp
#include <cstdint>
#include <vector>
#include <iostream>

std::vector<int64_t> dp;


void solve(int64_t n) {
    if (n == 1) return;

    auto check_memoization = [](int64_t n){
        if (dp[n] == -1) {
            solve(n);
        }
        return dp[n];
    };

    int64_t x1 = check_memoization(n - 1);
    int64_t x2 = n % 2 == 0 ? check_memoization(n / 2) : INT64_MAX;
    int64_t x3 = n % 3 == 0 ? check_memoization(n / 3) : INT64_MAX;

    dp[n] = std::min(std::min(x1, x2), x3) + n;
}

int main() {
    int n;
    std::cin >> n;

    std::vector<std::string> operation_types;
    dp.resize(n + 1, -1);
    dp[0] = INT64_MAX;
    dp[1] = 0;

    solve(n);

    std::cout << dp[n] << std::endl;
    while (n != 1) {
        if (dp[n - 1] + n == dp[n]) operation_types.push_back("-1"), n -= 1;
        else if (n % 2 == 0 && dp[n / 2] + n == dp[n]) operation_types.push_back("/2"),
             n /= 2;
        else if (n % 3 == 0 && dp[n / 3] + n == dp[n]) operation_types.push_back("/3"),
             n /= 3;
    }

    for (auto it = operation_types.begin(); it != operation_types.end(); ++it) {
        std::cout << *it;
        if (it < operation_types.end() - 1) std::cout << ' ';
```

```
46 ||     }
47 || }
```

3 Консоль

```
g++ descending.cpp
./a.out
82
202
-1 /3 /3 /3 /3%
```

4 Тест производительности

Сравним рекурсивное решение с мемоизацией и без:

```
g++ descending_no_memo.cpp
./a.out
500
1143
Recursion without memoization: 995ms
g++ descending.cpp
./a.out
500
1143
Recursion with memoization: 0ms
```

Как видно, решение с мемоизацией работает моментально, в то время как решение без мемоизации затрачивает целую секунду.

5 Выводы

Благодаря этой лабораторной работе я узнал, что такое динамическое программирование, как следует распознавать задачи на этот метод, а так же способы решения задач на динамическое программирование: восходящий и нисходящий анализ.

Список литературы

[1] *Динамическое программирование - ИТМО.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Динамическое_программирование`
(дата обращения: 20.09.2024).