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: Д. С. Ляшун
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б

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 объём 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.

Вариант: 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.

1 Описание

Требуется написать реализацию алгоритма решения поставленной задачи. Для этого воспользуемся следующей идеей динамического программирования: пусть массив *costs* хранит минимальную суммарную стоимость операций, которые необходимо выполнить для получения 1 от числа i. По умолчанию считаем, что $costs[1] = 0$, тогда значения в массиве *costs* в позициях от 2 до n могут быть найдены по следующему правилу:

$$costs[i] = min \begin{cases} 0 \mid i \leq 1 \\ costs[i-1] + i \mid i > 1 \\ costs[i \ / \ 2] + i \mid i > 1,\ i \vdots 2 \\ costs[i \ / \ 3] + i \mid i > 1,\ i \vdots 3 \end{cases}$$

Для вывода последовательности операций, приводящей к оптимальному ответу, можно завести дополнительный массив *samples*, в котором для каждого i будет записан соответствующий выбор операции, который использовался для получения оптимального ответа в i. Таким образом, используя записанные выборки оптимальных операций, можно совершить обратный проход от n до 1 и найти последовательность операций.

Оценим время работы алгоритма и его затраты по памяти. Массивы *costs* и *samples* длины n строятся в цикле по n итераций, вывод оптимальной последовательности операций производится также примерно за время, которое зависит от n. Таким образом, алгоритм работает за время $O(n)$ и потребляет $O(n)$ оперативной памяти.

2 Исходный код

Ниже приведён исходный код программы, в котором производится чтение входного числа n, последовательное построение массива с динамикой *costs* и выборками *samples*, по которым будет восстановлена последовательность операций, приводящая к оптимальному ответу:

```cpp
#include <iostream>
#include <vector>
int main() {
    int n;
    std::cin >> n;
    std::vector<int> costs(n + 1);
    std::vector<char> samples(n + 1);
    for (int i = 2; i <= n; ++i) {
        char choice = 1;
        costs[i] = costs[i - 1];
        if (i % 2 == 0 && costs[i] > costs[i / 2]) {
            costs[i] = costs[i / 2];
            choice = 2;
        }
        if (i % 3 == 0 && costs[i] > costs[i / 3]) {
            costs[i] = costs[i / 3];
            choice = 3;
        }
        costs[i] += i;
        samples[i] = choice;
    }
    std::cout << costs[n] << std::endl;
    int work = n;
    while (work != 1) {
        if (samples[work] == 1) {
            std::cout << "-1";
            work -= 1;
        }
        else if (samples[work] == 2) {
            std::cout << "/2";
            work /= 2;
        }
        else if (samples[work] == 3) {
            std::cout << "/3";
            work /= 3;
        }
        if (work != 1) {
            std::cout << " ";
        }
    }
    std::cout << std::endl;
}
```

3 Консоль

```
dmitry@dmitry-VirtualBox:~/Work_place/DA_labs/lab7$ g++ main.cpp -o solution
dmitry@dmitry-VirtualBox:~/Work_place/DA_labs/lab7$ ./solution
82
202
-1 /3 /3 /3 /3
dmitry@dmitry-VirtualBox:~/Work_place/DA_labs/lab7$ ./solution
303
617
/3 -1 /2 /2 -1 /3 /2 /2 -1
dmitry@dmitry-VirtualBox:~/Work_place/DA_labs/lab7$ ./solution
512
1022
/2 /2 /2 /2 /2 /2 /2 /2 -1
dmitry@dmitry-VirtualBox:~/Work_place/DA_labs/lab7$
```

4 Тест производительности

Тест производительности представляет из себя сравнение времени построения динамики посредством последовательного прохода по массиву и путём работы рекурсивной функции и мемоизации.

Значение входного числа n	Время работы динамики на массиве	Время работы рекурсивной функции
100	19 мкс.	18 мкс.
1000	163 мкс.	228 мкс.
10000	2020 мкс.	3646 мкс.
100000	22071 мкс.	44274 мкс.
1000000	195098 мкс.	Ошибка сегментирования
10000000	1741876 мкс.	Ошибка сегментирования

Как видно, время построения динамики с использованием рекурсивной функции и мемоизации заметно проигрывает построению посредством прохода по массиву, поскольку вызов функций является довольно время-затратной операцией ввиду работы со стеком вызовов, а также выделения памяти под каждую функцию.

Также можно увидеть, что на последних больших тестах алгоритм построения с использованием рекурсивной функции перестал работать и выдавал ошибку сегментирования, поскольку происходило переполнение стека вызовов (на Unix машинах его размер составляет всего 8 Мбайт). Таким образом, использовать алгоритм построения динамики посредством прохода по массиву предпочтительнее.

5 Выводы

Выполнив седьмую лабораторную работу по курсу «Дискретный анализ», я познакомился с основными методами динамического программирования и написал программу для решения поставленной задачи посредством работы с массивом динамики и последовательном вычислении значений для малых задач, среди которых затем выбиралось наиболее оптимальное решение для больших задач соответственно.

В ходе выполнения работы основную трудность для меня составил правильный вывод результата программы – в условии к задаче не было оговорено, что после вывода последовательности операций необходимо обязательно производить перевод на новую строку, из-за чего возникала ошибка представления на первых тестах.

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.