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. А. Терво
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б

Дата:
Оценка:
Подпись:

Москва, 2022

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.

Разработать программу на языке C или C++, реализующую построенный алгоритм. Формат входных и выходных данных описан в варианте задания:

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 $2 \leq n \leq 107$.

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

Требуется решить поставленную задачу медотом динамического программирования.

По сути, метод динамического программирования заключается в разбиении задачи на подзадачи, решения каждой по отдельности и объединения их решения для решения исходной задачи. Зачастую подзадачи получаются одинаковыми, в этом случае результат её решения запоминается и используется в дальнейшем, таким образом сокращается количество вычислений.

Для того, чтобы задачу можно было решить при помощи ДП, она должна обладать следующими свойствами [1]:

- перекрывающиеся подзадачи;
- оптимальная подструктура;
- возможность запоминания решения часто встречающихся подзадач.

2 Исходный код

Для решения задачи вычислим минимальную стоимость операций для всех x таких, что $x < n$.

Решаемая нами подзадача формулируется следущим образом: *Для всех* $i \in \{1..n-1\}$ *вычислить стоимость* $i+1$, $i*2$ *и* $i*3$ *при условии, что* $i*2 \leq n$ *и* $i*3 \leq n$.

Для хранения стоимости и проделанных операций создадим два вектора *dp* и *op*. В которых для каждого $x \in \{1..n\}$ в ячейке с индексом x хранятся минимальная стоимость его преобразования и операция, приводящая к текущему числу из предыдущего соответственно.

```
1  #include <iostream>
2  #include <vector>
3
4  using namespace std;
5
6  int main() {
7    int n;
8    cin >> n;
9    vector<long long> dp(n + 1, -1);
10   vector<int> op(n + 1);
11   dp[1] = 0;
12
13   for (int i = 1; i < n; ++i) {
14     if (dp[i + 1] == -1 || dp[i + 1] > dp[i] + i + 1) {
15       dp[i + 1] = dp[i] + i + 1;
16       op[i + 1] = 1;
17     }
18     if (i * 2 <= n && (dp[i * 2] == -1 || dp[i * 2] > dp[i] + i * 2)) {
19       dp[i * 2] = dp[i] + i * 2;
20       op[i * 2] = 2;
21     }
22     if (i * 3 <= n && (dp[i * 3] == -1 || dp[i * 3] > dp[i] + i * 3)) {
23       dp[i * 3] = dp[i] + i * 3;
24       op[i * 3] = 3;
25     }
26   }
27   printf("%lld\n", dp[n]);
28   int i = n;
29   while(i > 1) {
30     if (op[i] == 1) {
31       i = i - 1;
32       printf("-1 ");
33     }
34
35     if (op[i] == 2) {
```

```
36          i = i / 2;
37          printf("/2 ");
38        }
39
40        if (op[i] == 3) {
41          i = i / 3;
42          printf("/3 ");
43        }
44      }
45      printf("\n");
46    }
```

3 Консоль

```
[alext@alext-pc solution]$ ./solution
82
202
-1 /3 /3 /3 /3
[alext@alext-pc solution]$ ./solution
785
2382
-1 /2 /2 /2 /2 -1 /3 /2 /2 /2 -1
[alext@alext-pc solution]$ ./solution
92347278
165643249
/3 /2 -1 /3 /2 /2 -1 /2 /2 /2 -1 /2 /2 -1 /3 /2 /2 /2 /2 -1 /3 /2 -1 /3 -1
/3 /3 -1 /2 -1
```

4 Тест производительности

Сравнение будет производиться с жадным алгоритмом. Я придумал его простейшую реализацию, в которой к числу мы пытаемся применить операцию, максимально его сокращающую, т.е. если число кратно 3, делим на 3, если число чётное, делим на 2, если ничего выше нельзя сделать, вычитаем единицу.

Этот алгоритм не гарантирует правильный ответ.

```
[alext@alext-pc solution]$ ./solution
10000
DP time: 175us
[alext@alext-pc solution]$ ./bench
10000
Greedy algorithm time: 7us
[alext@alext-pc solution]$ ./solution
1000000
DP time: 16854us
[alext@alext-pc solution]$ ./bench
1000000
Greedy algorithm time: 7us
```

Видно, что жадный алгоритм сильно выигрывает у ДП за счёт того, что в нём выполняется намного меньше вычислений. Также не используется вектор для хранения всех вычисленных стоимостей, а вектор операций куда меньшего размера, из-за чего сокращается ещё и количество используемой памяти. Но при использованни жадного алгоритма в такой его реализации приходится жертвовать правильностью ответа.

5 Выводы

При выполнении этой лабораторной работы я изучил принципы и подходы динамического программирования, с которыми поверхностно знакомился ещё в школе.

Наиболее сложной частью решения задачи при помощи ДП для меня остаётся правильное определение подзадач, на которые нужно разбить исходную задачу.

Динамическое программирование бывает полезно для относительно несложного решения задач, которые без помощи ДП либо решаются сложными для понимания алгоритмами, либо решаются сложнее в смысле временной сложности, либо не решаются вообще.

Список литературы

[1] *Динамическое программирование — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Динамическое_программирование` (дата обращения: 16.12.2022).

[2] Список использованных источников оформлять нужно по ГОСТ Р 7.05-2008