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. Боглаев
Преподаватель: А. А. Кухтичев
Группа: М8О-306Б-22
Дата:
Оценка:
Подпись:

Москва, 2024

Лабораторная работа №7

Задача: Используя метод динамического программирования разработать алгоритм решения задачи, определяемой своим вариантом: оценить время работы алгоритма и объем затрачиваемой памяти. Перед выволнением задания необходимо обосновать пременимость метода динамического программирования.

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

Вариант 4: 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

Согласно [1]: «Динамическое программировние позволяет решать задачи, комбинируя решения вспомогательный задач. Каждая вспомогательная задача решается только один раз. Это позволяет избежать одних и тех же повторных вычислений каждый раз, когда встречается данная подзадача.»

Динамическое программирование, как правило, применяется к задачам оптимизации (optimization problems). В таких задачах возможно наличие многих решений. Каждому варианту решения можно сопоставить какое-то значение, и нам нужно найти среди них решение с оптимальным (минимальным или максимальным) значением.

Процесс разработки алгоритмов динамического программирования можно разбить на четыре перечисленных ниже этапа:

1. Описание структуры оптимального решения.
2. Рекурсивное определение значения, соответствующего оптимальному решению.
3. Вычисление значения, соответствующего оптимальному решению, с помощью метода восходящего анализа.
4. Составление оптимального решения на основе информации, полученной на предыдущих этапах.

Сохранение результатов выполнения функций для предотвращения повторных вычислений называется **мемоизацией**. Перед вызовом функции производится проверка: вызывалась функция или нет. Если функция вызывалась, то нужно взять результат ее вызова.

Задачей, где можно применить мемоизацию, является задача о нахождении числа Фибоначчи под номером n.

Также в динамическом программировании используются методы восходящего и нисходящего анализов для решения задачи. В восходящем методе сначала решаются простые задачи, а после более сложные. В нисходящем наоборот, сначала сложные, а после простые.

2 Исходный код

Нужно посчитать стоимость преобразования исходного числа в 1. Чтобы это сделать, нужно разработать алгоритм динамического программирования.

Есть три операции над числом: вычесть 1, разделить на 2, разделить на 3. Для составления алгоритма решения задачи используем метод восходящего анализа, вычислим стоимость преобразования для чисел, стоящих перед исходным числом.

Например, стоимость преобразования из числа 1 в 1 равно 0. Для числа 2 стоимость равна 2 и тд.

Создадим векстор длины $n + 1$, чтобы работать с числами от 1 до n. Далее для каждого числа, начиная с 2, вычислим стоимость преобразования в 1. Причем будем проверять, какую операцию использовать дешевле, используя предыдущие результаты.

Так мы сможем найти минимальную стоимость преобразования.

Также создадим вектор для сохранения операций, которые мы производим над числами. В функции $MakeSequence$ восстанавливаем последовательность операций, которые мы проделали для преобразования.

Для удобства создали пару $TPair$, чтобы вернуть из функции минимальную стоимость и послеловательность операций.

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>


using TPair = std::pair<int, std::vector<std::string>>;


std::vector<std::string> MakeSequence(int n, std::vector<std::string> &opers) {
    std::vector<std::string> seq;

    int cur_num = n;

    while (cur_num > 1) {
        seq.push_back(opers[cur_num]);

        if (opers[cur_num] == "-1") {
            cur_num -= 1;
        } else if (opers[cur_num] == "/2") {
            cur_num /= 2;
        } else if (opers[cur_num] == "/3") {
            cur_num /= 3;
        }
    }
```

```
26      return seq;
27  }
28
29
30  TPair MinCost(int n) {
31      std::vector<long long> dp(n + 1, 0);
32      std::vector<std::string> opers(n + 1, "");
33
34      for (int i = 2; i <= n; i++) {
35
36          dp[i] = dp[i - 1] + i;
37          opers[i] = "-1";
38
39          if (i % 2 == 0 && dp[i] > dp[i / 2] + i) {
40              dp[i] = dp[i / 2] + i;
41              opers[i] = "/2";
42          }
43
44          if (i % 3 == 0 && dp[i] > dp[i / 3] + i) {
45              dp[i] = dp[i / 3] + i;
46              opers[i] = "/3";
47          }
48      }
49
50      std::vector<std::string> seq = MakeSequence(n, opers);
51
52      return std::make_pair(dp[n], seq);
53  }
54
55
56  int main() {
57      int n;
58      std::cin >> n;
59      TPair res = MinCost(n);
60      std::cout << res.first << std::endl;
61
62      for (const std::string & str : res.second) {
63          std::cout << str << " ";
64      }
65
66      std::cout << std::endl;
67      return 0;
68  }
```

3 Консоль

```
alex@wega:~/$ ./main
12
18
/3 /2 -1
alex@wega:~/$ ./main
82
202
-1 /3 /3 /3 /3
alex@wega:~/$ ./main
100
213
/2 /2 -1 /3 /2 /2 -1
alex@wega:~/$ ./main
3421
8839
-1 /3 /3 /2 -1 /3 /3 /3 -1 /3 -1
alex@wega:~/$
```

4 Тест производительности

Сравним алгоритм с наивным алгоритмом. Тесты состоят из чисел: 82, 100, 200, 500.

```
alex@wega:~/$ ./a.out
82
Naive: 0.534 ms
DP: 0.107 ms
alex@wega:~/$ ./a.out
100
Naive: 0.742 ms
DP: 0.079 ms
alex@wega:~/$ ./a.out
200
Naive: 9.439 ms
DP: 0.038 ms
alex@wega:~/$ ./a.out
500
Naive: 641.927 ms
DP: 0.056 ms
```

Можно заметить, что динамическое программирование работает гораздо быстрее наивного алгоритма, потому что в наивном алгоритме стоимость вычисляется на каждом шаге, в то время как в динамическом программировании используется результат предыдущего шага.

5 Выводы

Выполнив седьмую лабораторную работу по курсу «Дискретный анализ», я изучил основные подходы динамического программирования, ознакомился с классическими задачами динамического прогнозирования и методами их решения. Реализовал алгоритм для своего варианта задания.

Динамическое программирования имеет множество применений. Его используют для оптимизации решения и в комбинаторных задачах. Также оно используется в задачах прогнозирования результатов и моделирования процессов.

Динамическое программирование является мощным инструментом для решения сложных задач, и его применение позволяет значительно улучшить производительность программных решений. Полученные знания и навыки будут полезны для дальнейшего изучения алгоритмов и разработки эффективных программных приложений.

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. (ISBN 5-8459-0857-4 (рус.))

[2] *Динамическое программирование — Викиконспекты.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=Динамическое_программирование`
(дата обращения: 09.11.2024).