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: Н. П. Ежов
Преподаватель: Н. С. Капралов
Группа: М8О-204Б
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 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.
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 описать метод динамического программирования, который позволит решить поставленную задачу. Итак, нам нужно уменьшить число n до 1 имея три операции ($-1/2/3$). Для того, чтобы быстро найти ответ построим массив всех ответов ans (массив состоит из элементов, которые содержат в себе мин. стоимость cost и следующий шаг operation), где по индексу i будет храниться минимальная стоимость превращения i в 1, а также одна из 3-х операций, которая является следующим шагом для уменьшения числа до 1. Находить ответы будем от меньшего к большему, т.е. мы начнём с 1 и будем идти до 10^7. Наименьшая стоимость на текущем шаге складывается из числа $i + min(ans[i-1].count, min(ans[i/3].count, ans[i/2].count))$. Естественно, чтобы проверить $ans[i/2].count$ и $ans[i/3].count$ мы сначала проверяем $i\%2 == 0$ и $i\%3 == 0$. Найдя минимум, прибавляем его к нашему текущему числу (это и будет минимальная стоимость для текущего числа), а операцию приравниваем в соответствии с минимумом (если минимальный $i-1$ то -1, если $i/2$ то $/2$, если $i/3$ то $/3$). После того, как мы подсчитали $ans[i]$ делаем $i++$, и начинаем алгоритм, описанный выше, сначала. Оценка построения массива ответов $O(3n)$, но т.к. константы не учитываются, то сложность будет $O(n)$. А время получения ответа для конкретного числа также не может быть больше $O(n)$ (нам приходиться идти по всему массиву до 1, т.к. для хранения всех операций для каждого числа потребуется слишком много памяти + активное копирование строк замедляет программу).

TNumber	
TNumber()	Конструктор для числа.
long long const	Минимальная стоимость преобразования числа в 1.
std::string *operation	Указатель на строку, содержащую в себе операцию, которая ведёт к минимальному пути.
bool operator <(TNumber &lhs, TNumber &rhs)	Оператор сравнения TNumber, который сравнивает поля cost.
std::ostream &operator « (std::ostream &of, TNumber &number)	Оператор вывода для TNumber (выводит операцию минимального пути).

TDynVector	
TDynVector() = default	Стандартный конструктор для массива ответов.
TDynVector(long long size)	Конструктор с указанием максимального числа n.
void Print(long long n)	Распечатать решение минимальное решение для числа n.
void PrintHelper(long long n)	Вспомагательная рекурсивная функция для печати оставшейся части решения.
TNumber& operator[](const long long index)	Оператор для получения элемента по индексу.
TDynVector()	Деструктор для массива ответов.
void CountCost()	Функция подсчёта решений для всех чисел $2 \leq i \geq 10^7$.
lonh lonh Size	Размер массива ответов.
TNumber *Data	Указатель на массив, где хранятся решения.
std::string empty	Пустая строка.
std::string fType = 1 "	Строка действия, указатель на которую будет хранится в TNumber.operation (также для "/2 "и "/3 ").

2 Исходный код

```
1  //
2  // struct.hpp
3  //
4  #include <string>
5  #include <iostream>
6  #include <vector>
7
8  struct TNumber{
9      long long cost;
10     std::string *operation;
11     TNumber(){
12         cost = 0;
13     }
14 };
15
16 bool operator <(TNumber &lhs, TNumber &rhs){
17     return lhs.cost < rhs.cost;
18 }
19
20 std::ostream &operator << (std::ostream &of, TNumber &number){
21     of << *number.operation;
22     return of;
23 }
24
25 class TDynVector{
26 private:
27     std::string empty;
28     std::string fType = "-1 ";
29     std::string sType = "/2 ";
30     std::string tType = "/3 ";
31     TNumber *Data;
32     long long Size = 0;
33     void CountCost(){
34         Data[1].operation = &empty;
35         long long currIndex = 2;
36         std::string *operation = &empty;
37         while(currIndex != Size + 1){
38             long long minCost = -1;
39             if(currIndex % 3 == 0){
40                 minCost = Data[currIndex/3].cost;
41                 operation = &tType;
42             }
43             if(currIndex % 2 == 0 && (Data[currIndex/2].cost < minCost || minCost ==
                   -1)){
44                 minCost = Data[currIndex / 2].cost;
45                 operation = &sType;
46             }
```

```
            if (currIndex - 1 > 0 && (Data[currIndex-1].cost < minCost || minCost ==
                -1)){
                minCost = Data[currIndex - 1].cost;
                operation = &fType;
            }
            Data[currIndex].cost = minCost + currIndex;
            Data[currIndex].operation = operation;
            ++currIndex;
        }
    }
public:
    TDynVector() = default;
    TDynVector(long long size){
        Data = new TNumber[size + 1];
        this->Size = size;
        this->CountCost();
    }
    void Print(long long n){
        std::cout << this->Data[n].cost << "\n";
        PrintHelper(n);
    }
    void PrintHelper(long long n){
        std::cout << this->Data[n];
        if(n != 1) {
            if (this->Data[n].operation == &tType) {
                PrintHelper(n / 3);
            } else if (this->Data[n].operation == &sType) {
                PrintHelper(n / 2);
            } else if (this->Data[n].operation == &fType) {
                PrintHelper(n - 1);
            }
        }
    }
    TNumber& operator[](const long long index){
        return Data[index];
    }
    ~TDynVector(){
        delete [] Data;
    }
};
```

```
//
// main.cpp
//
#include "struct.hpp"
int main(){
    std::cin.tie(nullptr);
    std::cout.tie(nullptr);
    std::ios_base::sync_with_stdio(false);
    long long n;
```

```
10      TDynVector dynVector(10000000);
11      if(std::cin >> n){
12          dynVector.Print(n);
13      }
14      std::cout << "\n";
15      return 0;
16  }
```

3 Консоль

```
nezhov@killswitch:~/CLionProjects/Diskran/lab7$ make
g++ -std=c++14 -O3 -Wextra -Wall -Werror -Wno-sign-compare -Wno-unused-result
-pedantic -o solution main.cpp
nezhov@killswitch:~/CLionProjects/Diskran/lab7$ ./solution
82
202
-1 /3 /3 /3 /3
```

4 Тест производительности

Сравнивать наш алгоритм будем с наивной реализацией, т.е. мы пройдёмся по всем путям от n до 1 и выберем из них минимальный.

```
nezhov@killswitch:~/CLionProjects/Diskran/lab7$ make benchmark
g++ -std=c++14 -O3 -Wextra -Wall -Werror -Wno-sign-compare -Wno-unused-result
-pedantic benchmark.cpp -o benchmark.o
nezhov@killswitch:~/CLionProjects/Diskran/lab7$ ./benchmark.o >temp
100
Dynamic programming solution time 80 ms
Naive algorithm solution time 0 ms
nezhov@killswitch:~/CLionProjects/Diskran/lab7$ ./benchmark.o >temp
500
Dynamic programming solution time 82 ms
Naive algorithm solution time 630 ms
nezhov@killswitch:~/CLionProjects/Diskran/lab7$ ./benchmark.o >temp
1000
Dynamic programming solution time 81 ms
Naive algorithm solution time 50940 ms
```

Глядя на результаты можно легко заметить одно из главных преимуществ динамического программирования - это почти неизменное время работы. Да, на совсем маленьком тесте динамическое программирование уступило наивному поиску, но чем больше был тест, тем больше была разница во времени.

5 Выводы

В ходе выполнения данной работы по дискретному анализу я познакомился с длинной арифметикой и реализовал её на практике. Также я познакомился с библиотекой GMP. Самым трудным было реализовать операцию деления длинных чисел. Также повлияло то, что прямое переписывание алгоритма деления [?] не помогло и пришлось самому придумывать способ выхода из данной ситуации.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 3-е издание.* — Издательский дом «Вильямс», 2013. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1328 с. (ISBN 5-8459-0857-4 (рус.)) (дата обращения: 20.04.2021).