Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №8 по курсу «Дискретный анализ»

Студент: И. О. Ильин
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-19

Дата:
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №8

Задача: Вариант №1

Разработать жадный алгоритм решения задачи, определяемой своим вариантом. Доказать его корректность, оценить скорость и объём затрачиваемой оперативной памяти.

Реализовать программу на языке C или C++, соответсвующую построенному алгоритму. Формат входных и выходных данных описан в варианте задания.

На первой строке заданы два числа, N и $p > 1$, определяющие набор монет некоторой страны с номиналами $p_0, p_1, ..., p_{N-1}$. Нужно определить наименьшее количество монет, которое можно использовать для того, чтобы разменять заданную на второй строчке сумму денег $M \leq 2^{32} - 1$ и распечатать для каждого i-го номинала на i-ой строчке количество участвующих в размене монет. Кроме того, нужно обосновать почему жадный выбор неприменим в общем случае (когда номиналы могут быть любыми) и предложить алгоритм, работающий при любых входных данных.

Формат входных данных

На первой строке заданы два числа N и $p > 1$, определяющие набор монет некоторой страны с номиналами $p_0, p_1, ..., p_{N-1}$. На второй сторке находится сумма денег которую необходимо разменять $M \leq 2^{32} - 1$.

Формат результата

Для каждого i-го номинала на i-ой строчке вывести количество участвующих в размене монет.

1 Описание

Требуется написать реализацию жадного алгоритма для решения задачи сдачи в монетах

Основная идея жадных алгоритмов состоит в том, что на каждом этапе решения подзадачи принимаются локально оптимальные решения, при этом конечное решение также окажется оптимальным [1].

Для решения данной задачи воспользуемся следующим алгоритмом. Поскольку имеющиеся монеты кратны числу p, то на каждом шаге можно брать максимальное количество монет, которые мы еще можем взять, не опасаясь, что вместо одной монеты p_i лучше было бы взять несколько монет p_{i-1}, посколько p_i и p_{i-1} кратны.

Таким образом, начиная с монет максимального номинала будем набирать максимальное их количество, после переходить к монетам меньших номиналов. Временная и пространственная сложности составят $O(M)$, где M – количество номиналов монет.

Однако, в общем случае (когда номиналы могут быть любыми) данный жадный алгоритм является неприменимым, поскольку нарушается правило, по которому выбирается локально оптимальное решение: взяв максимальное количество монет определенного номинала, мы не можем гарантировать оптимальность последующих решений.

Задачу такого типа нужно решать с помощью динамического программирования: для текущей суммы рассматриваем всевозможные способы взятия монеты, используя оптимальные решения для предыдущих сумм. Временная и пространственная сложности $O(M \cdot N)$, где M – количество номиналов монет, N – сумма для размена.

2 Исходный код

В файле *coins.h* объявим функцию *minCoinCharge* для поиска минимального количества монет.

```
#pragma once
#include <vector>
namespace NGreedyCoins {
    std::vector<long long> minCoinCharge(const long long coinsCount, const long long coinValue, long long toExchange);
} // /namespace NGreedyCoins
```

В файле *coins.cpp* напишем реализацию функции *minCoinCharge* в соответствии с раннее описанным алгоритмом.

```
#include "coins.h"
#include <cmath>
namespace NGreedyCoins {
    std::vector<long long> minCoinCharge(const long long coinsCount, const long long coinValue, long long toExchange) {
        std::vector<long long> result(coinsCount, 0);

        for (long long i = coinsCount - 1; i >= 0; --i) {
            long long currCoinValue = pow(coinValue, i);
            if (toExchange < currCoinValue) {
                continue;
            }

            result[i] = toExchange / currCoinValue;
            toExchange -= result[i] * currCoinValue;
            if (toExchange == 0) {
                break;
            }
        }

        return result;
    }
} //namespace NGreedyCoins
```

Также, в другом пространстве имён реализуем аналогичную функцию, которая будет реализовывать алгоритм решения той же задачи, но динамическим путём.

dp_coins.h

```
#pragma once
#include <vector>
namespace NDPCoins {
    std::vector<long long> minCoinCharge(const std::vector<long long> & coins, const long long toExchange);
} // namespace NDPCoins
```

dp_coins.cpp

```cpp
#include <limits>
#include "dp_coins.h"

namespace NDPCoins {
    std::vector<long long> minCoinCharge(const std::vector<long long> & coins, const long long toExchange) {
        std::vector<long long> dp (toExchange + 1, LLONG_MAX);
        std::vector<std::vector<long long> > coinsSet (
                toExchange + 1,
                std::vector<long long> (coins.size(), 0)
                );
        dp[0] = 0;

        for (long long currSum = 0; currSum <= toExchange; ++currSum) {
            for (size_t j = 0; j < coins.size(); ++j) {
                long long coin = coins[j];
                if (coin <= currSum) {
                    if (dp[currSum] >= dp[currSum - coin] + 1) {
                        dp[currSum] = dp[currSum - coin] + 1; //
                        coinsSet[currSum] = coinsSet[currSum - coin]; //
                        ++coinsSet[currSum][j];
                    }
                }
            }
        }
        return coinsSet[toExchange];
    }
} // namespace NDPCoins
```

3 Консоль

```
MacBook-Pro:da_lab_08 mr-ilin$ ./wrapper.sh
[2021-05-19 00:43:29] [INFO] Compiling...
g++ -std=c++17 -pedantic -Wall -Wno-unused-variable -c main.cpp -o main.o
g++ -std=c++17 -pedantic -Wall -Wno-unused-variable -c coins.cpp -o coins.o
g++ -std=c++17 -pedantic -Wall -Wno-unused-variable main.o coins.o -o solution
[2021-05-19 00:43:30] [INFO] Making unittest...
g++ -std=c++17 -pedantic -Wall -Wno-unused-variable -c unit_tests.cpp -o unit_tests.o
g++ -std=c++17 -pedantic -Wall -Wno-unused-variable -c dp_coins.cpp -o dp_coins.o
g++ -std=c++17 -pedantic -Wall -Wno-unused-variable unit_tests.o coins.o dp_coins.o
-o unit_test -lgtest -lgtest_main
[==========] Running 3 tests from 2 test suites.
[----------] Global test environment set-up.
[----------] 1 test from GreetyCoinsSuite
[ RUN      ] GreetyCoinsSuite.greetyMinCoinChargeCanonicalTest
[       OK ] GreetyCoinsSuite.greetyMinCoinChargeCanonicalTest (0 ms)
[----------] 1 test from GreetyCoinsSuite (0 ms total)

[----------] 2 tests from DPCoinsSuite
[ RUN      ] DPCoinsSuite.DpMinCoinChargeCanonicalTest
[       OK ] DPCoinsSuite.DpMinCoinChargeCanonicalTest (0 ms)
[ RUN      ] DPCoinsSuite.DpMinCoinChargeUsualTest
[       OK ] DPCoinsSuite.DpMinCoinChargeUsualTest (0 ms)
[----------] 2 tests from DPCoinsSuite (0 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 2 test suites ran. (0 ms total)
[  PASSED  ] 3 tests.
[2021-05-19 00:43:32] [INFO] Executing tests/t01.t...
OK
[2021-05-19 00:43:32] [INFO] No failed tests,hooray
MacBook-Pro:da_lab_08 mr-ilin$ ./solution
3 5
71
1
4
2
```

4 Тест производительности

Тест производительности представляет из себя сравнение двух описанных алгоритмов на случайных данных, подходящих для применения жадного алгоритма.

```
MacBook-Pro:da_lab_08 mr-ilin$ make bench
g++ -std=c++17 -pedantic -Wall -Wno-unused-variable -c benchmark.cpp -o benchmark.o
g++ -std=c++17 -pedantic -Wall -Wno-unused-variable -c coins.cpp -o coins.o
g++ -std=c++17 -pedantic -Wall -Wno-unused-variable -c dp_coins.cpp -o dp_coins.o
g++ -std=c++17 -pedantic -Wall -Wno-unused-variable benchmark.o coins.o dp_coins.o
-o benchmark
MacBook-Pro:da_lab_08 mr-ilin$ cat bench_test/01.t
77296 3
53
MacBook-Pro:da_lab_08 mr-ilin$ ./benchmark <bench_test/01.t
Greedy       4529 us
Dp     107861 us
MacBook-Pro:da_lab_08 mr-ilin$ cat bench_test/02.t
152609 5
1208
MacBook-Pro:da_lab_08 mr-ilin$ ./benchmark <bench_test/02.t
Greedy       9100 us
Dp    4729362 us
MacBook-Pro:da_lab_08 mr-ilin$ ./benchmark
10 5
10
Greedy          4 us
Dp         38 us
MacBook-Pro:da_lab_08 mr-ilin$ ./benchmark
10 5
10000
Greedy          4 us
Dp      12961 us
```

Как видно, жадный алгоритм выиграет с большим преимуществом у динамического, что объясняется раннее описанными временными сложностями данных алгоритмов. Особенно отставание заметно на последнем тесте: с увеличением суммы время выполнения динамического алгоритма увеличивается в разы

5 Выводы

В ходе выполнения данной лабораторной работы, я на практике познакомился с жадными алгоритмами, которые похожи на динамические, ведь также сводят вычисление большой задачи к маленьким подзадачам, однако для каждой подзадачи не ищут всевозможные решения, а берут только одно локально оптимальное, причём заведомо известно как его искать.

Также стоит учитывать применимость жадного алгоритма в контексте задачи на этапе планирования кода.

В данной задаче использование жадного алгоритма было оправдано, потому что имелись ограничения на набор входных данных (монеты не случайны, а кратны одному числу). Однако, по скорости выполнения и объёму занимаемой памяти он был в разы лучше универсального динамического алгоритма.

Список литературы

[1] *Жадные алгоритмы — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Жадный_алгоритм` (дата обращения: 10.05.2021).

[2] David Pearson. *A Polynimial-time Algorithm for the Change-Making Problem .*
URL: `https://graal.ens-lyon.fr/ abenoit/algo09/coins2.pdf` (дата обращения: 10.05.2021).